REAL-TIME TRAJECTORY GENERATION AND CONTROL FOR AUTONOMOUS VEHICLES

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Oliver Purwin

January 2009

REAL-TIME TRAJECTORY GENERATION AND CONTROL FOR AUTONOMOUS VEHICLES

Oliver Purwin, Ph.D.

Cornell University 2009

For many years robotics has provided solutions to problems which are either dull, dirty, or dangerous. Advancements in computer processing, sensors, and actuator technology made it possible for some of these robotic systems to require less and less human interaction. However, autonomy still poses many unsolved challenges. This dissertation presents a set of algorithms to control high-performance autonomous vehicles in real-time. All algorithms are designed with actual application in mind. They have been implemented and tested successfully on real land-based or airborne autonomous vehicles.

The first chapter describes an algorithm to calculate near-optimal minimum time trajectories for four wheeled omnidirectional vehicles, which can be used as part of a high-level path planner. The algorithm is based on a relaxed optimal control problem. It takes limited friction and vehicle dynamics into account, as encountered in high-performance omnidirectional vehicles. The low computational complexity makes the application in real-time feasible. An implementation of the algorithm on a real vehicle is presented and discussed.

The second chapter presents a cooperative decentralized path-planning algorithm for a group of autonomous agents that provides guaranteed collision-free trajectories in real-time. The algorithm is robust with respect to arbitrary delays in the wireless traffic, possible sources being transmission time and error correction. Agents move on reserved areas which are guaranteed not to intersect, therefore ensuring safety. A handshaking

procedure guarantees recent information states for the agents. Conflicts between agents are resolved by a cost-based negotiation process. The basic algorithm is augmented by the introduction of waypoints, which increase performance at the cost of additional wireless traffic. An implementation of the algorithm is tested in simulation and successfully applied to a real system of autonomous robots. The results are presented and discussed.

The third chapter presents an algorithm to iteratively perform an aggressive maneuver, i.e. drive a system quickly from one state to another. A simple model which captures the essential features of the system is used to compute the reference trajectory as the solution of an optimal control problem. Based on a lifted domain description of that same model an iterative learning controller is synthesized by solving a linear least-squares problem. The controller adjusts a feedforward signal using the results of experiments with the system. The non-causality of the approach makes it possible to anticipate recurring disturbances. Computational requirements are modest, allowing controller update in real-time. The experience gained from successful maneuvers can be used to adjust the model, which significantly reduces transients when performing similar motions. The algorithm is successfully applied to a real quadrotor unmanned aerial vehicle. The results are presented and discussed.

Biographical Sketch

Oliver Christoph Purwin was born on October 7, 1975 in Frankfurt, Germany. He began studying Mechanical Engineering at the Darmstadt University of Technology (TUD), Darmstadt, Germany, in 1996. In 2000/2001 he took part in a merit-based exchange program with Cornell University. He graduated 1st in his class with a Master of Engineering in Mechanical Engineering. During that time he joined the interdisciplinary RoboCup team, which competed in an international robotic soccer competition in Seattle in Summer of 2001, winning 3rd place. In 2002 he graduated from TUD with a Diplom-Ingenieur in Mechanical Engineering. In the Fall of 2002 he returned to Cornell University in order to pursue a Ph.D. in Mechanical Engineering with a concentration in Dynamics and Control and a Minor in Computer Science. He continued to be associated with the RoboCup team, which won 1st place in 2003 in Padua, Italy, and 2nd place in 2005 in Osaka, Japan. His responsibilities included prediction and controls algorithms.

To my parents, who made all of this possible.

To Yashoda, for her love and patience.

To Dave “Bones” Schneider, a true friend.

The Road goes ever on and on
    Down from the door where it began.
Now far ahead the Road has gone,
    And I must follow, if I can,
Pursuing it with eager feet,
    Until it joins some larger way
Where many paths and errands meet.
    And whither then? I cannot say.

- J.R.R. Tolkien

ACKNOWLEDGEMENTS

During my time at Cornell I had a lot of help and support from many people, some of whom I would like to mention explicitly.

First my thanks to my advisor, Professor Raffaello D'Andrea, who gave me the great opportunity to work with some awesome toys, and whose guidance and aim for perfection certainly made me a better engineer. I also wish to thank the members of my Special Committee, Professors Mark Campbell and Hod Lipson, for answering my questions whenever I had any.

My friends Yashoda, Betsy, Mike, the Star Trek crew: Dave "Bones", Ken "Captain Kirk", Jeremy "Scotty", and Luke. Thank you just for being in there for the ride.

I would also like to thank the MAE administrative staff, especially Marcia Sawyer, who takes excellent care of her "children".

Finally, my gratitude to Pedro, Simone, and π, who provided a home away from home, when I needed one.

Table of Contents

List of Tables

List of Figures

PREFACE

Advancements in computer processing, sensors, and actuator technology enable an increasing amount of autonomy in deployed vehicles as seen in industrial robotics, drones for battlefield reconnaissance, and the Mars rovers. Autonomous vehicles allow the operator to shift his focus from menial control tasks to supervision of the high-level behavior. For example, this makes it possible for a single operator to control multiple vehicles at the same time.

The theme of this dissertation is the control of high-performance autonomous vehicles in real-time. It provides approaches to three common problems encountered with the deployment of autonomous vehicles: How to move a single vehicle? How to move many vehicles? How to accomplish a dynamically challenging maneuver? The presented controls algorithms are founded on the vehicle dynamics and are designed with actual application in mind. They are implemented in a computationally efficient manner and tested successfully on real land-based or airborne autonomous vehicles.

The first chapter describes an algorithm to move a single vehicle. The algorithm calculates near-optimal minimum time trajectories for four wheeled omnidirectional vehicles, which can be used as part of a high-level path planner. Given the initial position and velocity of the vehicle and the desired destination the algorithm computes a trajectory that the vehicle is able to follow. The dynamics of the vehicle including friction between wheels and the ground are taken into account to determine the maximum possible acceleration in each direction. The acceleration envelope is simplified, which decouples the degrees of freedom of the vehicle. Solving an optimal control problem allows the computationally efficient assembly of the solution from a set of trajectory building blocks. The low computational complexity allows the application in real-time. The algorithm was motivated by the annual RoboCup competition and applied successfully to the small-size league during the years 2003 through 2005.

The second chapter extends the task from moving a single vehicle to controlling a fleet of vehicles. Each vehicle or agent is acting independently, having decentralized intelligence and control. However, they are able to communicate with each other via wireless communication. The task of each agent is to reach a certain destination without colliding with each other. A cooperative decentralized path-planning algorithm is presented that provides guaranteed collision-free trajectories in real-time. The algorithm is a safety protocol which is robust with respect to arbitrary delays in the wireless traffic, possible sources being transmission time, error correction, or network failure. The underlying idea is that the agents move on reserved areas which are guaranteed not to intersect, therefore ensuring safety. A handshaking procedure guarantees recent information states for the agents. Conflicts between agents are resolved by a cost-based negotiation process. The basic algorithm is augmented by the introduction of waypoints, which increase performance at the cost of additional wireless traffic. In order for this algorithm to work the agents have to meet a few modest requirements: The agents have to be able to localize themselves, communicate with each other, and they have to use a deterministic motion primitive, i.e., when moving to a particular destination they have to know the approximate path ahead of time. A valid motion primitive is the trajectory generation presented in chapter 1. An implementation of the algorithm has been tested in simulation and successfully applied to a real system of autonomous robots.

The third chapter touches on the subject of accomplishing a single aggressive motion, for example a quick turn. The algorithm presented here complements the trajectory generation from chapter 1 in the sense that it pushes the envelope of what a vehicle is capable of doing, while the trajectory generation sacrifices some performance for ease of computation. A method is proposed to iteratively drive a system quickly from one state to another. A simple model which captures the essential features of the system is used to compute the reference trajectory as the solution of an optimal control problem.

The system is linearized and discretized about the reference trajectory and the resulting dynamics are written into a single large matrix. Based on this lifted domain description an iterative learning controller is synthesized by solving a linear least-squares problem. The controller adjusts a feedforward signal using the results of experiments with the system. This approach is non-causal, since the controller has access to future dynamics and is able to anticipate recurring disturbances. Computational requirements are modest, allowing controller update in real-time. The experience gained from successful maneuvers can be used to adjust the model, which significantly reduces transients when performing similar motions. The algorithm is successfully applied to the autonomous flying vehicle (AFV), a quadrotor which has been designed and built at Cornell University.

The main contributions of this dissertation are:

- Derivation of a novel trajectory generation algorithm for a particular class of high-performance omnidirectional vehicles, including implementation in Matlab and C++.
- Presentation of a real-time safety protocol for a fleet of decentralized agents which prevents collisions in the presence of arbitrary wireless delay.
- Successful application of a least-squares based iterative learning controller to an autonomous rotorcraft. Reliable tracking of short open-loop maneuvers using a relatively simple model.

Chapter 1
Trajectory Generation and Control for Four Wheeled Omnidirectional Vehicles

Abstract

This paper describes an algorithm to calculate near-optimal minimum time trajectories for four wheeled omnidirectional vehicles, which can be used as part of a high-level path planner. The algorithm is based on a relaxed optimal control problem. It takes limited friction and vehicle dynamics into account, as encountered in high-performance omnidirectional vehicles. The low computational complexity makes the application in real-time feasible. An implementation of the algorithm on a real vehicle is presented and discussed.

1.1 Introduction

Omnidirectional vehicles have some desirable properties: They are very maneuverable, able to navigate tight quarters, and have few constraints on path planning. The small-size league of the annual RoboCup competition is an example of a highly dynamic environment where omnidirectional vehicles have been employed extremely successfully since 2000 [6] [7].

Path planning in general is a difficult task, especially when considering vehicle dynamics and moving obstacles. Rapidly changing environments require a re-computation of the path in real-time. There are many approaches to solve this problem, which are based on different assumptions about the hardware and the environment. Paromtchik and Rembold [11] and Muñoz et al. [12], for example, used a sequence of splines to generate a path which includes waypoints. The splines contained time information, so that the

vehicle's desired velocity could be limited depending on the used hardware. Faiz and Agrawal [13] approximated the set of all feasible states of the system with polytopes, which took the dynamics and other constraints into account. Moore and Flann [8] presented a trajectory generation algorithm for an off-road vehicle. The basis for the path generator was a set of mission goals that had to be achieved. They used an A* algorithm to determine trajectories as a combination of steps, ramps, decaying exponentials, and sinusoidal functions. Watanabe et al. [9] used a resolved acceleration approach on their omnidirectional robotic platform. They inverted the dynamics of the system and implemented a PI or PD controller with a feedforward term to minimize the error between desired and achieved trajectory. Liu et al. [14] implemented a method called trajectory linearization control (TLC), which is based on linearization along the desired trajectory and inversion of the dynamics. Kalmár-Nagy et al. [5] developed a trajectory generation algorithm which computed a minimum time path based on the dynamics of the vehicle and the motor characteristics.

With the recent advancements in robot hardware, some of the basic assumptions for generating optimal paths have changed. Robots can accelerate at much higher rates than they used to. Some robots are no longer power but friction limited, since the drive motors can deliver enough torque to make the wheels slip even at high velocities. Furthermore, due to the high accelerations the effect of weight transfer becomes more significant. Weight transfer can cause the normal forces between the wheels and the ground to change, thus altering the available amount of traction and the maximum acceleration.

The objective of this paper is to present a trajectory generation algorithm for high-performance omnidirectional vehicles. The algorithm computes the minimum time trajectory from a given initial state to a given final state while taking limited friction and weight transfer into account. The output is a sequence of velocities for the vehicle,

which have to be tracked by low-level control. This algorithm can readily be used as part of a high-level path planning algorithm, for example as the obstacle-free guidance system in [10].

The algorithm is designed for vehicles with four powered wheels moving on a plane surface. These vehicles are overactuated, which means that different combinations of control inputs can have the same net effect on the system. The problem of overactuated vehicles is discussed in [17] and several techniques are given of how to allocate the control efforts in order to cause the desired system response.

In the derivation of the presented algorithm ideal control of the actuators is assumed, i.e., motor torques can be chosen arbitrarily within the physical limits. On a real vehicle the wheel forces would be determined by a dedicated controller [6], which is beyond the scope of this paper.

This paper is organized in the following way: Section 1.2 describes the assumptions made and covers the derivation of the vehicle equations of motion. In Section 1.3 the model of the vehicle dynamics is simplified and the rotational and translational degrees of freedom (DOF) are decoupled. The result is an acceleration profile, which is independent of the vehicle orientation. Section 1.4 presents a solution to the relaxed optimal control problem, i.e., finding a minimum time solution to drive the system to the desired final destination given the previously derived vehicle characteristics. Section 1.5 describes the performance of the algorithm in simulation, while Section 1.6 covers results from the implementation on a real vehicle of the Cornell RoboCup system. Section 1.7 concludes the paper.

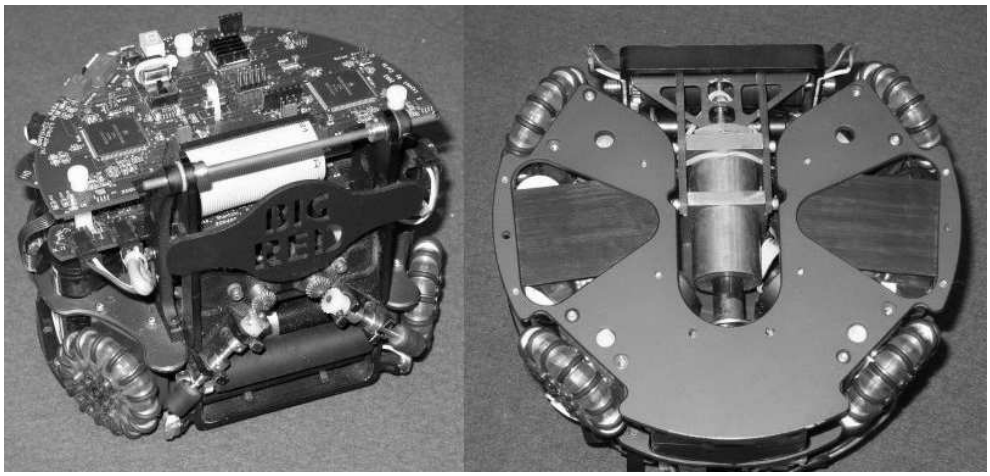

Figure 1.1: 2003 Cornell RoboCup robot (left), robot wheelbase (right)

1.2 Vehicle Dynamics

The basis for the following derivations is a four wheeled omnidirectional vehicle, see Figure 1.1. The drive modules are equally spaced at 90 degrees. The center of mass (CM) is assumed to be exactly above the geometrical center of the drive system.

1.2.1 Motor Characteristics

The main assumption here is that the acceleration of the vehicle is friction limited, which means that the maximum acceleration can be achieved over the entire velocity range. This is a reasonable approximation for vehicles which are equipped with strong drive motors and do not have much room to accelerate or decelerate, as discussed in [6].

1.2.2 Friction Force and Weight Transfer

Friction has been modelled as Coulomb friction. For more sophisticated friction models, see Olsson et al. [15] for example. The maximum acceleration force a wheel

can exert is $f_a = \mu n$, where n is the normal force between the wheel and the ground, and μ is the coefficient of friction. In general, the normal force n is not only a function of vehicle mass and geometry but it depends on the current acceleration vector of the vehicle. This effect is called weight transfer [1]. It means that during an acceleration phase the weight distribution on the wheels changes due to the inertia of the robot mass. For example, when accelerating in the forward direction the normal force of the front wheels is reduced while at the same time the rear wheels are loaded more heavily.

1.2.3 Derivation of Equations of Motion

The first step is to derive the equations of motion which govern the vehicle's dynamics. Figure 1.2 depicts the free-body diagram of the vehicle. The global coordinate

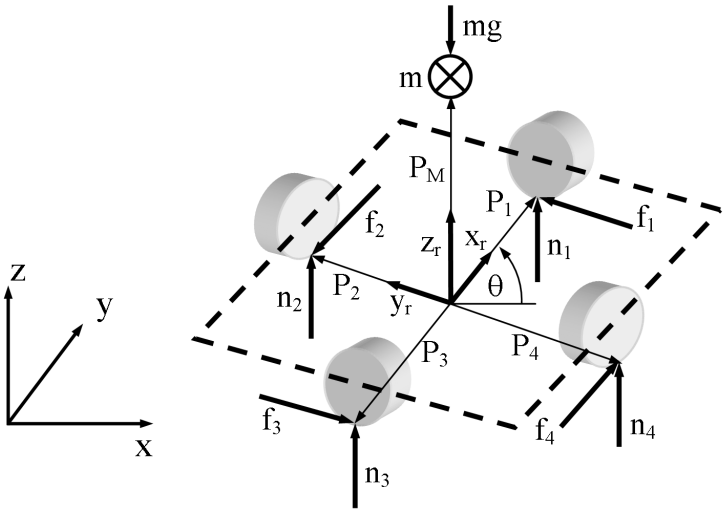

Figure 1.2: Free body diagram

system is defined by x, y, and z. The vehicle frame of reference is defined by x_r, y_r, and z_r. The angle θ is the rotation of the vehicle in the $x-y$ plane, i.e., it is the rotation of the local coordinates with respect to the global coordinate system. The mass of the vehicle is m. The forces n_i are the normal forces between the wheels and the ground, the forces

f_i are the friction forces. The positions of the wheels with respect to the CM are defined by the vectors $\mathbf{P}_i$:

$$\mathbf{P}_1 = l\begin{bmatrix}1\\0\\0\end{bmatrix}, \quad \mathbf{P}_2 = l\begin{bmatrix}0\\1\\0\end{bmatrix}, \quad \mathbf{P}_3 = l\begin{bmatrix}-1\\0\\0\end{bmatrix}, \quad \mathbf{P}_4 = l\begin{bmatrix}0\\-1\\0\end{bmatrix} \tag{1.1}$$

The parameter l describes the distance from the geometrical center to the wheels. The driven directions $\mathbf{D}_i$ of the wheels are orthogonal to the position vectors $\mathbf{P}_i$.

$$\mathbf{D}_1 = \begin{bmatrix}0\\1\\0\end{bmatrix}, \quad \mathbf{D}_2 = \begin{bmatrix}-1\\0\\0\end{bmatrix}, \quad \mathbf{D}_3 = \begin{bmatrix}0\\-1\\0\end{bmatrix}, \quad \mathbf{D}_4 = \begin{bmatrix}1\\0\\0\end{bmatrix} \tag{1.2}$$

The CM of the vehicle is at

$$\mathbf{P}_\mathrm{M} = h\begin{bmatrix}0\\0\\1\end{bmatrix} \tag{1.3}$$

The rotation matrix $\mathbf{R}(\theta)$ relates the local (vehicle) frame of reference (FOR) to the global (Newtonian) FOR:

$$\mathbf{R}(\theta) = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix} \tag{1.4}$$

Taking the force and moment balance in the global FOR yields two sets of equations that define the vehicle dynamics:

$$\begin{bmatrix}m\ddot{x}\\ m\ddot{y}\\ m\ddot{z}\end{bmatrix} = \mathbf{R}(\theta)\left(\sum_i f_i\mathbf{D}_i + \sum_j n_j\begin{bmatrix}0\\0\\1\end{bmatrix} + mg\begin{bmatrix}0\\0\\-1\end{bmatrix}\right) \tag{1.5}$$

$$\begin{bmatrix}J_x\ddot{\theta}_x\\ J_y\ddot{\theta}_y\\ J\ddot{\theta}\end{bmatrix} = \mathbf{R}(\theta)\left(\sum_i (-\mathbf{P}_\mathrm{M} + \mathbf{P}_i)\times f_i\mathbf{D}_i + \sum_j \left(-\mathbf{P}_\mathrm{M} + \mathbf{P}_j\right)\times n_j\begin{bmatrix}0\\0\\1\end{bmatrix}\right) \tag{1.6}$$

where J is the moment of inertia. Subscripts $\bullet_x$ and $\bullet_y$ indicate rotation about the coordinates x and y respectively. At this point the assumption is made that the normal forces are always positive, i.e., the vehicle does not tip. Thus

$$\ddot{z} = \ddot{\theta}_x = \ddot{\theta}_y = 0 \tag{1.7}$$

The equations of motion in the x - y plane are

$$m\ddot{x} = \cos\theta(f_4 - f_2) - \sin\theta(f_1 - f_3) \tag{1.8}$$

$$m\ddot{y} = \sin\theta(f_4 - f_2) + \cos\theta(f_1 - f_3) \tag{1.9}$$

$$J\ddot{\theta} = l\sum_i f_i \tag{1.10}$$

where the wheel forces f_i can be arbitrarily chosen within the limits posed by the maximum friction forces

$$|f_i| \leq f_{i,max} = \mu n_i \tag{1.11}$$

The acceleration envelope $\mathcal{A}_0$ is defined as the boundary of the set of all feasible combinations of $\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$. Within the envelope, every combination of $\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$ can be achieved by the vehicle.

In order to find the acceleration envelope of the vehicle, (1.8) through (1.10) have to be solved in terms of the friction forces f_i, which are functions of the normal forces n_i. Therefore, expressions for the normal forces have to be found first. Equations (1.5) and (1.6) only yield 6 equations for the 7 unknowns. In addition to the equations of motion it is assumed that the normal forces are distributed as follows

$$n_1 + n_3 = n_2 + n_4 \tag{1.12}$$

For a derivation of (1.12), see Appendix A.

In order to find explicit expressions for the normal forces n_i, the moment balances (1.6) have to be pre-multiplied by $\mathbf{R}^{-1}(\theta)$. The inverse of $\mathbf{R}(\theta)$ always exists, since $\mathbf{R}(\theta)$

is non-singular for all $\theta \in [0, 2\pi]$ [3]. With assumption (1.7) this leads to

$$f_1 h - f_3 h + n_2 l - n_4 l = 0 \tag{1.13}$$

$$f_2 h - f_4 h - n_1 l + n_3 l = 0 \tag{1.14}$$

The system of equations consisting of (1.12), (1.13), (1.14), and the force balance in the z direction from (1.5) is solved for n_i in order to yield

$$\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{bmatrix} = \frac{1}{4l} \begin{bmatrix} 2h(f_2 - f_4) + lmg \\ 2h(f_3 - f_1) + lmg \\ 2h(-f_2 + f_4) + lmg \\ 2h(-f_3 + f_1) + lmg \end{bmatrix} \tag{1.15}$$

At this point the control inputs u_i are introduced, which are a measure of how much torque the motors provide.

$$f_i = u_i f_{i,\max}, \quad u_i \in [-1, 1] \tag{1.16}$$

Equations (1.11) and (1.15) are substituted into (1.16), which leads to implicit expressions for f_i. At the same time, the terms $f_2 - f_4$ and $f_3 - f_1$ are replaced by acceleration terms from (1.8) and (1.9):

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \frac{m\mu}{4l} \begin{bmatrix} u_1(-2h(\ddot{x}\cos\theta + \ddot{y}\sin\theta) + lg) \\ u_2(-2h(-\ddot{x}\sin\theta + \ddot{y}\cos\theta) + lg) \\ u_3(2h(\ddot{x}\cos\theta + \ddot{y}\sin\theta) + lg) \\ u_4(2h(-\ddot{x}\sin\theta + \ddot{y}\cos\theta) + lg) \end{bmatrix}, \quad u_i \in [-1, 1] \tag{1.17}$$

The goal is to find expressions for $\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$ which are only functions of u_i, μ, m, h, l, g, and θ. In order to achieve this, (1.17) is substituted back into the equations of motion (1.8), (1.9), and (1.10). The result is a system of coupled differential equations, which is only dependent upon accelerations and the control efforts. All terms containing normal

or friction forces have been eliminated.

$$\begin{aligned}
\ddot{x} &= \frac{\mu}{4l}[lg((u_4 - u_2)\cos\theta - (u_1 - u_3)\sin\theta) + \\
&\quad 2h(\ddot{x}(u_1 + u_3 - u_2 - u_4)\cos\theta\sin\theta + \\
&\quad \ddot{y}((u_2 + u_4)\cos^2\theta + (u_1 + u_3)\sin^2\theta))] \qquad (1.18)\\
\ddot{y} &= \frac{\mu}{4l}[lg((u_4 - u_2)\sin\theta + (u_1 - u_3)\cos\theta) + \\
&\quad 2h(\ddot{x}(-(u_2 + u_4)\sin^2\theta - (u_1 + u_3)\cos^2\theta) + \\
&\quad \ddot{y}(-u_1 - u_3 + u_2 + u_4)\cos\theta\sin\theta)] \qquad (1.19)\\
\ddot{\theta} &= \frac{\mu m}{4J}[lg(u_1 + u_2 + u_3 + u_4) + \\
&\quad 2h(\ddot{x}((u_3 - u_1)\cos\theta + (u_2 - u_4)\sin\theta) + \\
&\quad \ddot{y}((u_4 - u_2)\cos\theta + (u_3 - u_1)\sin\theta))] \qquad (1.20)
\end{aligned}$$

Solving (1.18) through (1.20) explicitly for $\ddot{x}$, $\ddot{y}$, and $\ddot{\theta}$ yields

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\theta} \end{bmatrix} = \mathbf{R}(\theta)\frac{lg\mu}{4l^2 + h^2\mu^2(u_2 + u_4)(u_1 + u_3)}$$

$$\begin{bmatrix} l(u_4 - u_2) + 0.5\mu h(u_1 - u_3)(u_2 + u_4) \\ l(u_1 - u_3) + 0.5\mu h(u_2 - u_4)(u_1 + u_3) \\ m/J(h^2\mu^2 \sum_i \frac{u_1u_2u_3u_4}{u_i} + l^2 \sum_i u_i) \end{bmatrix}, u_i \in [-1, 1] \qquad (1.21)$$

These nonlinear equations in u_i describe the set of admissible accelerations of the vehicle in the global frame of reference. The acceleration envelope is discussed in more detail in the next section.

1.3 Simplification of the Acceleration Envelope

Since (1.21) is nonlinear in u_i, a numerical approach is chosen in order to find the envelope. Discretizing the control efforts u_i, solving the equations using sample param-

eters, and plotting the results yields a discretized set of admissible accelerations. This approximation can be arbitrarily close to the continuous envelope, depending on the resolution of the discretization. Table 1.1 holds the parameters for the omnidirectional vehicle depicted in Figure 1.1, which is also used in Section 1.6 to show the implementation of the trajectory generation on an actual vehicle. Figure 1.3 shows the envelope

Table 1.1: Sample values for an omnidirectional vehicle

Parameter	μ	g	m	J	l	h
Value	0.8	9.81 m/s^2	2.7 kg	0.0085 m^{2}kg	0.08 m	0.05 m

for the sample case with $\theta = 0$. Depending on the given parameters the dimensions vary, but the characteristic shape is always the same. The resulting acceleration enve-

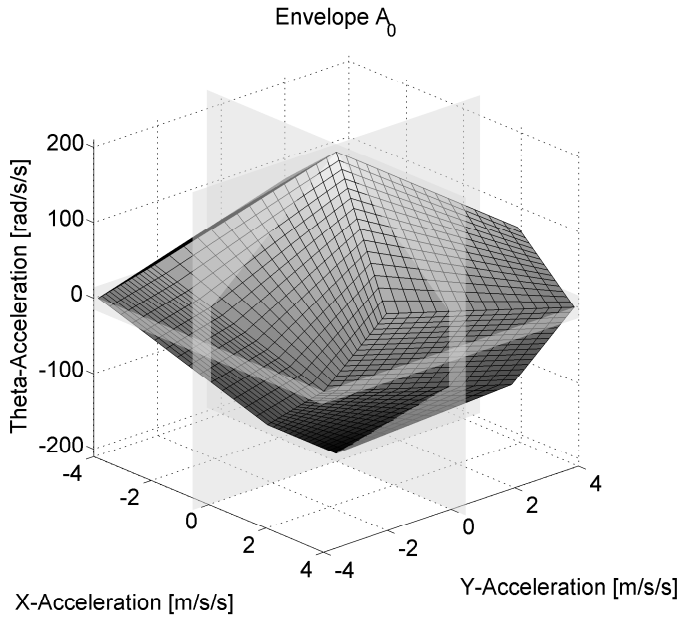

Figure 1.3: Acceleration envelope $\mathcal{A}_0$

lope $\mathcal{A}_0$ is not rotationally symmetric, which means that the maximum magnitude of the acceleration is determined by the direction of the acceleration vector. This dependency

makes it harder to compute the vehicle's minimum time trajectory. In order to find a closed form solution, the acceleration envelope is restricted, similarly to [5]. By taking the intersection of the acceleration envelopes for all directions θ, the influence of the orientation is removed:

$$\mathcal{Q} = \bigcap_{\theta \in [0,2\pi]} \mathbf{R}(\theta)\mathcal{A}_0 \tag{1.22}$$

$\mathcal{Q}$ defines the admissible set of accelerations that the vehicle can achieve, independent of the current orientation θ. The result for the sample envelope is depicted in Figure 1.4. Within $\mathcal{Q}$, any arbitrary combination of the three acceleration components can be

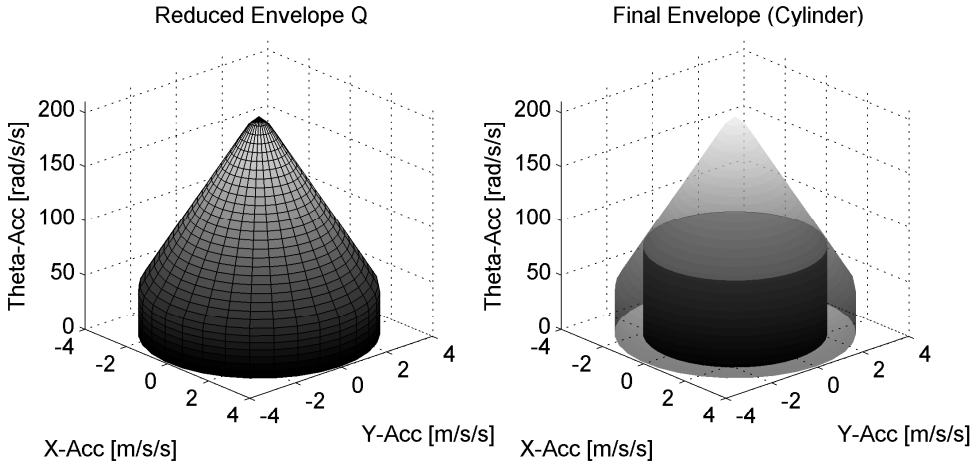

Figure 1.4: Reduced envelope $\mathcal{Q}$ (left), final cylindrical envelope (right)

chosen. It should be noted that the accelerations are coupled through the control efforts u_i. In order to make the optimal control problem in Section 1.4 faster to solve for real-time applications, the envelope is further simplified by decoupling the rotational DOF θ from the two linear DOFs. This is achieved by imposing a limit on the rotational and translational accelerations: $|\ddot{\theta}| \leq \ddot{\theta}_{\max}$ and $\sqrt{\ddot{x}^2 + \ddot{y}^2} \leq a_{\max}$, where $a_{\max}$ is the maximum linear acceleration at $\ddot{\theta} = \ddot{\theta}_{\max}$. This reduces the envelope to a cylinder with radius $a_{\max}$, as illustrated in Figure 1.4. The presented algorithm therefore assumes independent

control of translation and rotation, as described in [6]. However, the algorithm can easily be extended to not making this simplification.

In the following section, the resulting optimal control problem for the simplified envelope is posed and a solution is presented.

1.4 The Optimal Control Problem

For the remainder of this paper the vehicle rotation will be neglected, since it can be treated as a simplification of the translational cases. The objective is to find the minimum time path for the two linear DOFs, x and y, from a given initial state to a desired final state. The dynamics have been simplified to a double integrator subject to limitations on the acceleration and velocity. The state of the system consists of positions and velocities. The final velocity is always chosen to be zero in order to avoid discontinuities in the solutions when approaching the desired final state, see [2] [5]. It should be noted that when applying the algorithm in practice the vehicle hardly ever slows down to zero velocity since the destination will be changed continually depending on the environment [6, 7].

By definition, all accelerations inside the acceleration envelope can be achieved. With the simplifications made in Section 1.3 individual wheel forces are no longer required to describe the system dynamics. The inputs to the simplified system are the accelerations in x and y directions instead. The allocation of torques to the wheels is the task of a separate controller, which is not within the scope of this paper.

The optimal control problem is to minimize the time t_f for the system

$$\ddot{x}(t) = q_x(t) \tag{1.23}$$

$$\ddot{y}(t) = q_y(t) \tag{1.24}$$

with initial and final conditions

$$x(0) = 0, \quad x(t_\mathrm{f}) = x_\mathrm{f}, \quad \dot{x}(0) = \dot{x}_0, \quad \dot{x}(t_\mathrm{f}) = 0 \tag{1.25}$$

$$y(0) = 0, \quad y(t_\mathrm{f}) = y_\mathrm{f}, \quad \dot{y}(0) = \dot{y}_0, \quad \dot{y}(t_\mathrm{f}) = 0 \tag{1.26}$$

subject to constraints on the control effort and the state

$$\sqrt{q_x^2(t) + q_y^2(t)} \leq a_\mathrm{max} \tag{1.27}$$

$$\sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \leq v_\mathrm{max} \tag{1.28}$$

where $q_x(t)$ and $q_y(t)$ are the control efforts in x and y directions, t_f is the execution time, x_f and y_f are the final positions, $\dot{x}_0$ and $\dot{y}_0$ are the initial velocities. The maximum acceleration a_max is the radius of the cylindrical envelope from the previous section (see Figure 1.4), the maximum velocity v_max is defined by the motor specifications.

In order to make the problem more tractable, each DOF is handled independently at first. In the end, the final times of both DOFs are synchronized. Introducing a general DOF w, the problem can be written as:

$$\ddot{w}(t) = q_w(t) \tag{1.29}$$

$$w(0) = 0, \quad w(t_\mathrm{f}) = w_\mathrm{f}, \quad \dot{w}(0) = \dot{w}_0, \quad \dot{w}(t_\mathrm{f}) = 0 \tag{1.30}$$

$$|\dot{w}(t)| \leq v_{w,\mathrm{max}}, \quad |q_w(t)| \leq a_{w,\mathrm{max}} \tag{1.31}$$

The minimum time solution to this problem occurs on the boundary of the velocity and/or acceleration constraints [2]. This means that at any time the vehicle is following one of three strategies:

- accelerating: $q_w(t) = a_{w,\mathrm{max}}$
- decelerating: $q_w(t) = -a_{w,\mathrm{max}}$
- cruising: $|\dot{w}(t)| = v_{w,\mathrm{max}}, \quad q_w(t) = 0$

In order to find a complete solution the problem can be broken down into a combination of several distinct cases. Each case represents a possible state or condition of the system. These conditions are mutually exclusive, at any given time the system is in one and only one of these cases, depending on $\dot{w}_0$, w_f, $v_{w,\max}$, and $a_{w,\max}$. Each case has a control effort associated with it (e.g. case 1: $q_w(t) = a_{w,\max}$). The strategy is to apply this control effort until the conditions for a different case are satisfied or the final destination is reached with zero final velocity. The complete solution is therefore a sequence of cases.

The problem is being normalized to $w_\mathrm{f} \geq 0$ by inverting the signs of w_f and $\dot{w}_0$ if w_f is negative. Thus, the possible cases are reduced to the five cases shown in Figure 1.5. Case

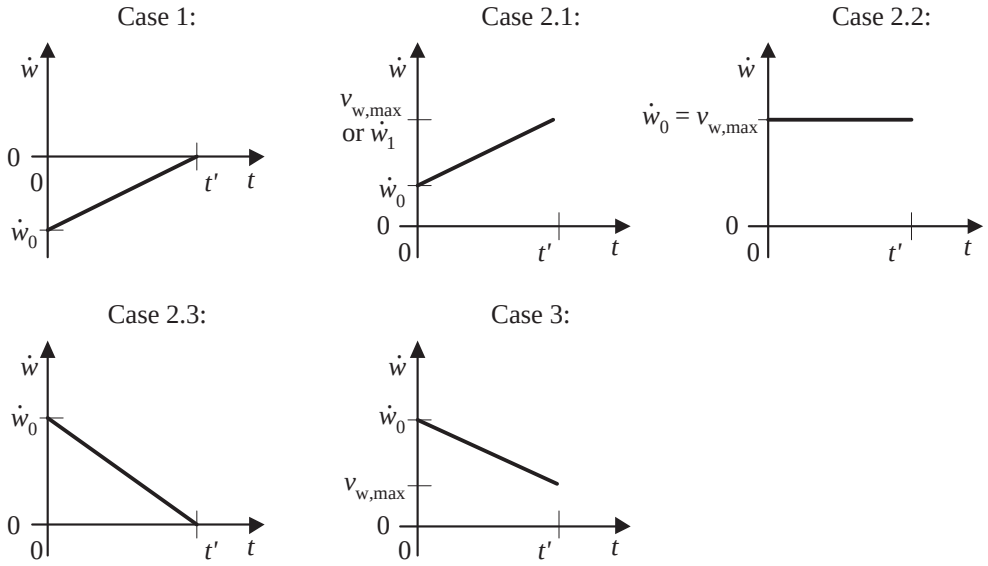

Figure 1.5: Possible optimal control cases

1 covers all initial conditions where $\dot{w}_0 < 0$: the vehicle is initially moving away from the destination and has to accelerate with $q_w(t) = a_{w,\max}$ in order to reverse direction. Case 3 is applicable if $\dot{w}_0 > v_{w,\max}$. The vehicle is moving too fast and has to decelerate with maximum control effort. This case is possible since $v_{w,\max}$ is not necessarily a hard physical constraint, but rather a desired maximum velocity which should not be

exceeded. Furthermore, the constraint $v_{w,\max}$ can be decreased artificially as part of the synchronization process of the final times, see below. For all other instances there are three choices left (acceleration, deceleration, coasting) depending on how far the destination is and how fast the vehicle moves initially. These three choices are covered by cases 2.1, 2.2, and 2.3. A possible sequence could look like:

> The vehicle is far away from the destination and moving slowly towards it. It will accelerate until it reaches $v_{w,\max}$ (case 2.1), cruise at $v_{w,\max}$ (case 2.2), and finally decelerate such that it reaches the destination with exactly zero final velocity (case 2.3).

Given the applied control effort and the initial conditions, it is possible to find a closed form solution for the final state of each case. The following list contains the requirements for each particular case, the applied control effort $q_w(t)$ associated with that case, the execution time t', the travelled distance $w' = w(t')$, and the final velocity $\dot{w}' = \dot{w}(t')$.

Case 1: $\dot{w}_0 < 0$

The initial velocity is negative, the vehicle has to accelerate with maximal control effort until it reaches $\dot{w}(t) = 0$.

$$q_w(t) = a_{w,\max}, \quad t' = -\frac{\dot{w}_0}{a_{w,\max}} \tag{1.32}$$

$$w' = \dot{w}_0 t' + \frac{a_{w,\max}}{2} t'^2 = -\frac{\dot{w}_0^2}{2a_{w,\max}}, \quad \dot{w}' = 0 \tag{1.33}$$

Case 2.1: $v_{w,\max} > \dot{w}_0 \geq 0 \quad \text{AND} \quad w_\mathrm{f} > \frac{\dot{w}_0^2}{2a_{w,\max}}$

The vehicle has to accelerate, either because the destination is far away or the initial velocity is small. This case has two subcases: In subcase I, case 2.1 is being followed by case 2.2, the vehicle reaches $v_{w,\max}$ and is cruising. In subcase II, case 2.1 is being

followed by case 2.3 and the vehicle decelerates until it reaches the final destination, see Figure 1.6. Define t_1 as the time when the vehicle has to start decelerating in order to

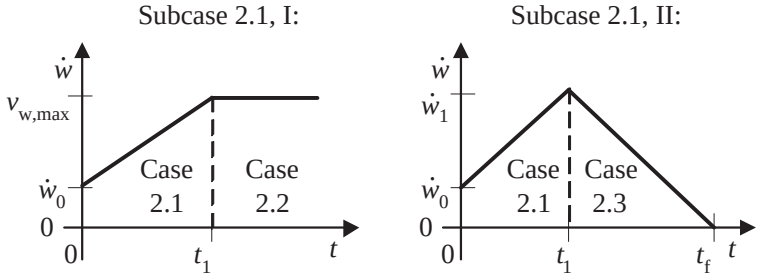

Figure 1.6: The subcases of case 2.1

avoid overshooting the destination. Also define $w_1 = w(t_1)$ and $\dot{w}_1 = \dot{w}(t_1)$. It follows that

$$\begin{aligned} w_1 &= \dot{w}_0 t_1 + \frac{a_{w,\max} t_1^2}{2}, \quad t_1 = \frac{\dot{w}_1 - \dot{w}_0}{a_{w,\max}} \\ &= \dot{w}_0 \frac{\dot{w}_1 - \dot{w}_0}{a_{w,\max}} + \frac{(\dot{w}_1 - \dot{w}_0)^2}{2a_{w,\max}} \qquad (1.34) \\ w_f - w_1 &= \frac{\dot{w}_1^2}{2a_{w,\max}} \qquad (1.35) \end{aligned}$$

Adding (1.34) and (1.35) and solving for $\dot{w}_1$ yields:

$$\dot{w}_1 = \sqrt{w_f a_{w,\max} + \frac{\dot{w}_0^2}{2}} \qquad (1.36)$$

where $\dot{w}_1$ has to be positive by definition. The decision, which of the two subcases is applicable, is based on a comparison of the time t_I to reach $v_{w,\max}$ and the time t_{II} when the vehicle has to decelerate.

$$t_I = \frac{v_{w,\max} - \dot{w}_0}{a_{w,\max}}, \quad t_{II} = \frac{\dot{w}_1 - \dot{w}_0}{a_{w,\max}} \qquad (1.37)$$

If $t_I < t_{II}$ then the vehicle reaches $v_{w,\max}$ and

$$\begin{aligned} q_w(t) &= a_{w,\max}, \quad t' = t_I \qquad (1.38) \\ w' &= \dot{w}_0 t' + \frac{a_{w,\max}}{2} t'^2 = \frac{v_{w,\max}^2 - \dot{w}_0^2}{2a_{w,\max}}, \quad \dot{w}' = v_{w,\max} \qquad (1.39) \end{aligned}$$

otherwise it has to brake before it reaches $v_{w,\max}$ and

$$q_w(t) = a_{w,\max}, \quad t' = t_{\mathrm{II}} \tag{1.40}$$

$$w' = w_1 = \frac{w_\mathrm{f}}{2} + \frac{\dot{w}_0^2}{2a_{w,\max}}, \quad \dot{w}' = \dot{w}_1 \tag{1.41}$$

Case 2.2: $\dot{w}_0 = v_{w,\max}$ AND $w_\mathrm{f} > \frac{\dot{w}_0^2}{2a_{w,\max}}$

The vehicle is cruising at maximum velocity until it has to decelerate.

$$q_w(t) = 0, \quad t' = \frac{w_\mathrm{f}}{v_{w,\max}} - \frac{v_{w,\max}}{2a_{w,\max}} \tag{1.42}$$

$$w' = w_\mathrm{f} - \frac{v_{w,\max}^2}{2a_{w,\max}}, \quad \dot{w}' = v_{w,\max} \tag{1.43}$$

Case 2.3: $v_{w,\max} \geq \dot{w}_0 > 0$ AND $w_\mathrm{f} \leq \frac{\dot{w}_0^2}{2a_{w,\max}}$

The vehicle has to decelerate until it reaches zero final velocity.

$$q_w(t) = -a_{w,\max}, \quad t' = \frac{\dot{w}_0}{a_{w,\max}} \tag{1.44}$$

$$w' = \frac{\dot{w}_0^2}{2a_{w,\max}}, \quad \dot{w}' = 0 \tag{1.45}$$

Case 3: $\dot{w}_0 > v_{w,\max}$

The vehicle moves faster than the allowed maximum velocity. This can be caused by the iterative solution procedure presented below. The vehicle has to decelerate until it reaches the allowed velocity.

$$q_w(t) = -a_{w,\max}, \quad t' = \frac{\dot{w}_0 - v_{w,\max}}{a_{w,\max}} \tag{1.46}$$

$$w' = \frac{1}{2a_{w,\max}}(\dot{w}_0^2 - v_{w,\max}^2), \quad \dot{w}' = v_{w,\max} \tag{1.47}$$

The procedure to find the complete solution is as follows:

1. Define initial and final states, set $t = 0$

   Normalize: $\dot{w}_0 = \text{sign}(w_\text{f})\dot{w}_0, \quad w_\text{f} = \text{sign}(w_\text{f})w_\text{f}$

2. Check which case is applicable, based on the initial conditions

3. Compute: $q_w(t)$, t', w', and $\dot{w}'$

4. Set: $t = t + t', \quad w_\text{f} = w_\text{f} - w', \quad \dot{w}_0 = \dot{w}'$

   Normalize: $\dot{w}_0 = \text{sign}(w_\text{f})\dot{w}_0, \quad w_\text{f} = \text{sign}(w_\text{f})w_\text{f}$

5. If the destination is not reached with zero final velocity: Go to 2

6. Total time to destination $t_{\text{f},w} = t$

The result of this algorithm is a minimum time trajectory for a single DOF. Given a particular instance of the problem (1.23) through (1.28), the solutions in x and y will yield different execution times $t_{\text{f},x}$ and $t_{\text{f},y}$ in general. In order to get the minimum time to destination for the vehicle the solutions have to be synchronized. This is done by adjusting the maximum allowed control effort and velocity for both DOFs via the parameter $\alpha \in (0, \pi/2)$:

$$a_{x,\max} = a_\max \cos\alpha, \quad v_{x,\max} = v_\max \cos\alpha \tag{1.48}$$

$$a_{y,\max} = a_\max \sin\alpha, \quad v_{y,\max} = v_\max \sin\alpha \tag{1.49}$$

Equations (1.48) and (1.49) satisfy the constraints (1.27) and (1.28). If either $x_\text{f} = 0$ or $y_\text{f} = 0$ with zero initial velocity, no synchronization is needed. Thus the exclusion of 0 and $\pi/2$ from α. The execution times $t_{\text{f},x}$ and $t_{\text{f},y}$ are continuous and strictly monotonously increasing/decreasing functions of α [18]. Therefore it is possible to use a bisection algorithm to find α:

1. Initial guess: $\alpha = \pi/4$, $\alpha_\min = 0$, $\alpha_\max = \pi/2$

2. Find minimum time trajectories for both x and y coordinates, given α

3. If $| t_{f,x} - t_{f,y} |$ is sufficiently small, keep the solutions and stop the search
4. If $(t_{f,x} > t_{f,y})$, set $\alpha = (\alpha - \alpha_{min})/2$, $\alpha_{max} = \alpha$
5. If $(t_{f,x} < t_{f,y})$, set $\alpha = (\alpha_{max} - \alpha)/2$, $\alpha_{min} = \alpha$
6. Go back to step 2

Using bisection the difference between the two execution times can be made arbitrary small. An alternative is to keep the number of iterations constant, effectively limiting the search time in a real-time application. The optimal value α_{opt} is defined as the α for which $t_{f,x} = t_{f,y}$. The difference between α_{opt} and α is bounded by

$$| \alpha - \alpha_{opt} | \leq \frac{\pi}{2^{N+1}} \tag{1.50}$$

where N is the number of iterations of the bisection. Therefore, the order of the search algorithm is $O(\log_2 N)$. It should be noted that the solution depends continuously on the initial and final conditions, i.e., small changes in the initial and final conditions only cause small changes in the solution. This property makes the solution robust to disturbances and noise.

1.5 Performance of the Algorithm in Simulation

This section describes how the implementation of the algorithm performs. One important aspect is the computation time. The recursive algorithm presented in the previous chapter was implemented in C++ and tested on a Pentium 4 (1.7 GHz clock speed). A Monte Carlo simulation yielded average computation times of 94 μs, with a standard deviation of 28 μs. Note that there are guarantees on the maximum number of operations per iteration, since a sequence can only consists of a maximum of five cases in a row.

Another point is the quality of the solutions of the proposed algorithm in terms of the execution time t_f. The derivation of the algorithm involves several simplifications and assumptions, which greatly reduce the required computational effort but at the same time increase the execution times of the found trajectories. The solution of the proposed algorithm $t_{\mathrm{f,approx}}$ is compared against two different benchmarks. Both are computed using RIOTS [61], an optimal control toolbox written in Matlab and C. The Matlab code for the simulation can be found at the author's web site [64]. It should be noted that parts of the simulation can only be run in conjunction with the RIOTS engine which is a commercial product and therefore not included.

RIOTS can solve optimal control problems with constraints on the state and the control effort. The first benchmark is the execution time $t_{\mathrm{f,full}}$ of the full problem, i.e., (1.21) subject to the velocity constraint (1.28). The second benchmark is the execution time $t_{\mathrm{f,2D}}$ of the two-dimensional problem (1.23) to (1.28). The parameters of the sample vehicle (Figure 1.1, Table 1.1) are used in all simulations. The maximum velocity and acceleration are limited to $v_{\max} = 2.0$ m/s and $a_{\max} = 3.92$ m/s^2. The velocity limit is required to maintain the friction limit assumption. The maximum acceleration is found by limiting the rotational acceleration to $\ddot{\theta}_{\max} = 44.9$ rad/s^2 and determining the intersection with the envelope (see Figure 1.4). If the rotational acceleration was any larger it would reduce the maximum allowed translational acceleration.

A Monte Carlo simulation was performed by generating random initial conditions and computing $t_{\mathrm{f,approx}}$, $t_{\mathrm{f,full}}$, and $t_{\mathrm{f,2D}}$. Figure 1.7 shows the simulation results. The abscissa depicts the ratio r_{tf} of the execution times $t_{\mathrm{f,full}}/t_{\mathrm{f,approx}}$ or $t_{\mathrm{f,2D}}/t_{\mathrm{f,approx}}$ respectively. The ordinate shows the fraction n_i/n_{tot}, where n_{tot} is the total number of solutions. The variable n_i is the number of solutions for which the execution time ratio $t_{\mathrm{f,RIOTS}}/t_{\mathrm{f,approx}}$ is smaller or equal than r_{tf}. That means that a fraction of n_i/n_{tot} of all solutions have a

ratio of final times of r_{tf} or better. Summarizing, the abscissa depicts the quality of the results, while the ordinate shows what fraction of the solutions achieves that quality.

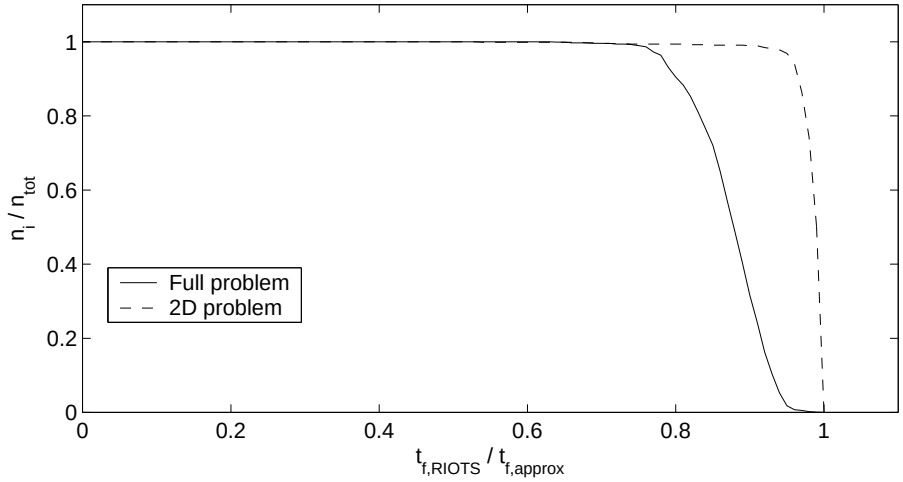

Figure 1.7: Comparison of execution times

As expected, the solution of the full problem yielded the smallest execution times, since it used the unreduced acceleration envelope. Due to the simplifications and relaxations the execution times $t_{\mathrm{f,approx}}$ are longest, but not by much. More than 94 % of $t_{\mathrm{f,approx}}$ are within 96 % of $t_{\mathrm{f,2D}}$. When comparing to the solutions of the full problem, more than 85 % of $t_{\mathrm{f,approx}}$ are within 82 % of $t_{\mathrm{f,full}}$. On the other hand, the reduction in computation time is significant: On the same computer the average RIOTS solution (both the full and the 2D solution) took more than 2 minutes while the simplified algorithm was executed in about 100 μs.

Figure 1.8 shows a comparison of a RIOTS trajectory (using the full envelope) and the trajectory generated by the presented algorithm. Plotted are x versus y positions of the following representative example:

$$x(0) = 1.143 \text{ m}, \quad x(t_{\mathrm{f}}) = 0.0 \text{ m}, \quad \dot{x}(0) = 0 \text{ m/s}, \quad \dot{x}(t_{\mathrm{f}}) = 0 \text{ m/s} \tag{1.51}$$

$$y(0) = 0.5 \text{ m}, \quad y(t_{\mathrm{f}}) = 0.0 \text{ m}, \quad \dot{y}(0) = -1.0 \text{ m/s}, \quad \dot{y}(t_{\mathrm{f}}) = 0 \text{ m/s} \tag{1.52}$$

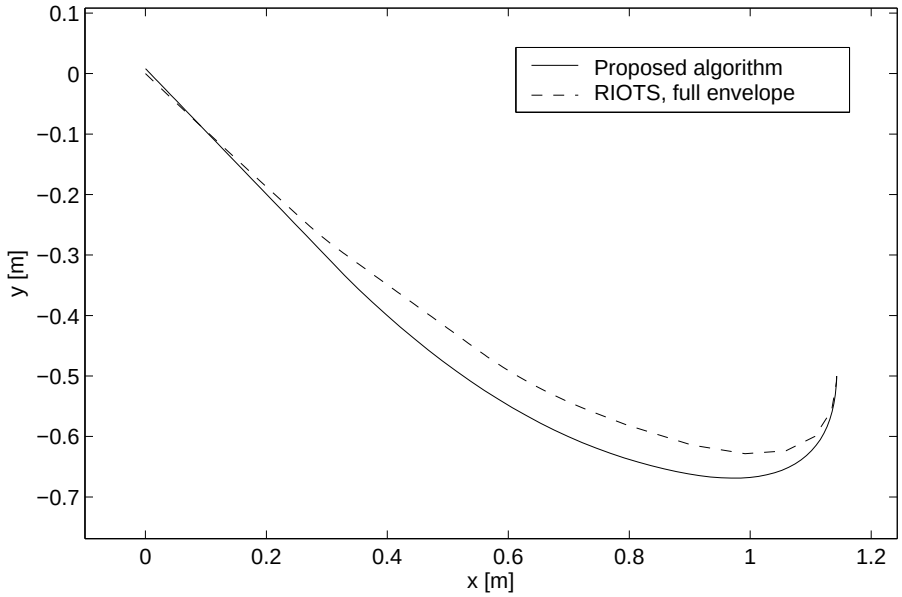

Figure 1.8: Comparison of Trajectories

1.6 Implementation

The new trajectory generation algorithm was implemented on the Cornell RoboCup system. RoboCup is a game of completely autonomous robotic soccer. The main sensor of the system is an overhead camera which takes pictures of the playing field at a rate of 60 Hz. The vision system analyzes these frames and determines the robots' positions and velocities. This data is passed on to the strategy module, which makes the decisions where to move the robots. Trajectory generation then computes minimum time paths, which are recomputed every frame in order to give immediate feedback. The velocity commands corresponding to these paths are sent to the robots. The robots track the velocity commands using essentially PI controllers for the wheel speeds.

In order to test the performance of the proposed trajectory generation algorithm, a robot was commanded to move along a line from A to B, using trajectory generation.

The rotation was held fixed at $\theta = 0$ rad. When the robot crossed a particular x coordinate x_c, the final destination was changed to C, so that the robot had to alter its course while moving. Four different situations were tested, with $x_{c,1} = -0.6$ m, $x_{c,2} = -0.2$ m, $x_{c,3} = 0.2$ m, and $x_{c,4} = 0.6$ m. Table 1.2 contains the destinations, Figure 1.9 depicts the results. Video clips of the vehicle executing the test pattern can be found at [64]. The Figure shows two paths for each x_c. The solid lines stand for the paths actually taken by the robots. The trajectories and vehicle commands are recomputed every frame to compensate for process and sensor noise. The dashed lines are the paths that were computed in the frame when the destination was changed from B to C.

Table 1.2: Destinations for implementation test

Destination	x-Coord	y-Coord
A	-1.0 m	-0.5 m
B	1.0 m	-0.5 m
C	0.0 m	0.5 m

The theoretical and actual paths deviate at most by about 0.1 m. This might seem a large error in comparison to the length of the entire trajectory, but it should be kept in mind that the vehicle is not tracking the dashed trajectories for more than one frame. The trajectories are being recomputed every frame which means that errors enter only in form of slightly changed initial conditions. On the other hand, for a short time horizon (≈ 0.3 s) the deviations between precomputed and actually taken trajectories are about an order of magnitude smaller. The errors are sufficiently small for effective obstacle avoidance and ball control in the small-size league of the RoboCup competition, where the algorithm was very successfully applied [6].

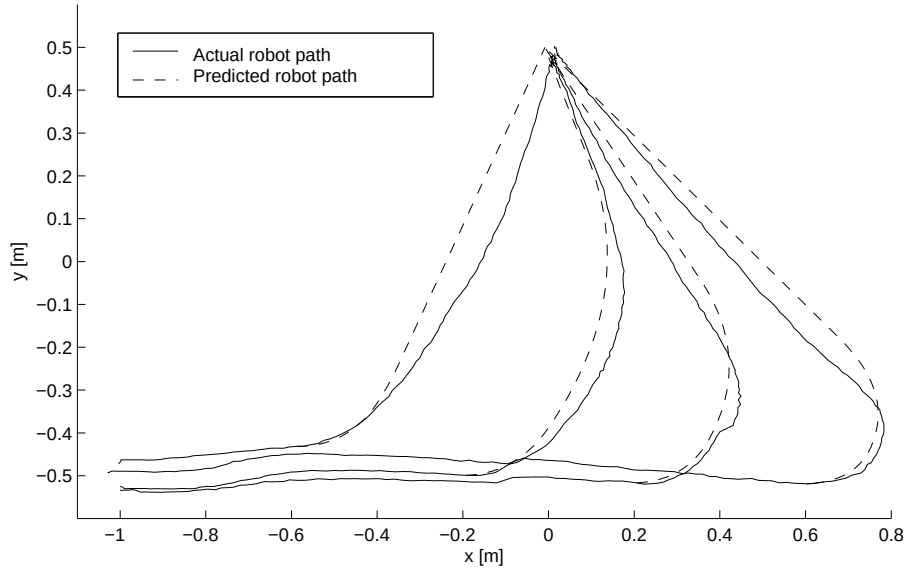

Figure 1.9: Implementation on Robot

1.7 Conclusion

A trajectory generation algorithm for omnidirectional vehicles has been presented. The algorithm takes the vehicle dynamics, limited friction, and weight transfer into account. It is tailored to high-performance vehicles that are mainly friction limited. It offers a computationally efficient way to calculate minimum time trajectories, which are close to the optimal solutions. In order to prove the feasibility of the concept, the algorithm was successfully applied to a real vehicle of the Cornell RoboCup system.

During the derivation of this algorithm certain assumptions were made about the location of the center of mass and the wheel positions. In practice, these locations are subject to manufacturing tolerances and therefore not perfectly well known. These deficiencies could be the topic of further investigations. In case that the CM is not centered, for example, additional terms are introduced in (1.6).

During the derivation of the vehicle dynamics it was assumed that the motors could

always provide sufficient torque to make the wheels slip. In reality, this assumption will break down for most vehicles at high speeds. An interesting approach would be to combine limited friction with a motor model such as presented in [5]. This would mean that at low speeds the motors can provide enough torque τ to make the wheels slip (τ_{fric}), but if the wheel velocity v gets larger than a certain threshold the maximum torque drops linearly, see Figure 1.10.

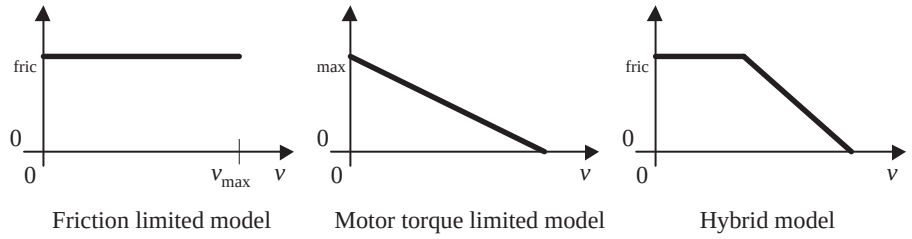

Figure 1.10: Torque-speed graphs

Also, this paper could be extended to non-zero final velocities. This would require finding means of handling the discontinuities that arise when approaching the final destination. There still remains great potential for future research in the area of trajectory generation for omnidirectional vehicles.

Chapter 2
Theory and Implementation of Path Planning by Negotiation for Decentralized Agents

Abstract

This paper presents a cooperative decentralized path-planning algorithm for a group of autonomous agents that provides guaranteed collision-free trajectories in real-time. The algorithm is robust with respect to arbitrary delays in the wireless traffic, possible sources being transmission time and error correction. Agents move on reserved areas which are guaranteed not to intersect, therefore ensuring safety. A handshaking procedure guarantees recent information states for the agents. Conflicts between agents are resolved by a cost-based negotiation process. The basic algorithm is augmented by the introduction of waypoints, which increase performance at the cost of additional wireless traffic. An implementation of the algorithm is tested in simulation and successfully applied to a real system of autonomous robots. The results are presented and discussed.

2.1 Introduction

Unmanned autonomous agents are being employed in an ever growing number of areas, such as military applications, disaster relief, exploration, and industrial mobile robots. A frequently occurring scenario is the dispatch of a group of agents, each agent moving towards a task location in order to perform a certain objective. While moving to their respective destinations agents have to avoid stationary and moving obstacles, for example the other dispatched agents.

This problem is typically encountered in the area of automatic air traffic conflict detection and resolution, but can be applied to land- or sea-based vehicles as well. Kuchar

and Yang [20] provide an overview of recently proposed methods. Non-cooperative methods are in general dealing with worst-case scenarios [21] while cooperative algorithms involve information exchange between the agents.

Cooperative approaches can be further broken down into centralized and decentralized methods. Centralized algorithms usually encompass the optimization of a global cost function [22], the solution being computed using, for example, semi-definite programming [23] or non-linear dynamic programming [24]. The strength of these methods lies in the optimality of the solution. However, gathering all information at a central location might be a challenging task in practice if the system is large.

In general, decentralized methods scale better with respect to the number of agents and are more robust since they do not possess a single point of failure [25]. An example for a cooperative decentralized method is presented by Schouwenaars et al. [26], an approach based on model predictive control (MPC). The optimal control problem includes vehicle dynamics and is solved using mixed-integer linear programming (MILP). Every agent is allotted a time slot in which to compute a dynamically feasible and guaranteed collision-free path. Inalhan et al. [27] recast the global optimization problem as several local problems, which are then iteratively solved by the agents in a decentralized fashion. The agents exchange information during each iteration in order to facilitate the solution process.

Other decentralized approaches include rule-based methods [28] and potential fields [29] [30]. Potential field methods allow the inclusion of vehicle dynamics and can be computationally fast, but in general they provide no guarantees of collision avoidance.

The algorithm presented in this paper is a decentralized path planner for a group of n agents, where every agent is trying to reach a task location while avoiding collisions

with each other. The task locations are chosen by some kind of high-level directive, for example the output of a task allocation problem [31]. Every agent is assumed to move according to dynamically feasible motion primitives. The approach is cooperative, since agents exchange their intentions and negotiate their paths via wireless communication. The entire algorithm is executed in real-time. There is a trade-off between the amount of wireless communication and performance: The higher the performance, the more communication is needed. One important property of this approach is robustness to arbitrary wireless latency, possible sources being transmission time, bandwidth limitations, error detection/recovery, or temporary network breakdown. The agents' paths are guaranteed collision-free, even if inter-agent communication is delayed by an arbitrary amount. Further, the algorithm scales well with respect to the number of agents. It is fully distributed, i.e., there is no need for a central data repository, clock, or arbitrator. Wireless communication is assumed to be provided by an ad-hoc network, for example [32]. The use of waypoints allows the agents to move around choke points thereby improving performance at the expense of more communication.

The decentralized nature of the algorithm in combination with safety guarantees, modelling of wireless traffic, and real-time execution makes this work an original contribution to the field.

The algorithm is based on the idea of every agent reserving an exclusive area for itself and remaining inside that area at all times. The agents can always come to a full stop inside the reserved areas in case problems arise, such as delayed wireless communication. No two reserved areas are allowed to intersect at any time, which guarantees safety. This fail-safe mechanism is similar in spirit to the passive collision avoidance described in [33]. The algorithm is implemented using computational geometry as in [34], which allows for fast processing. Reserved areas are based on the particular dy-

namics of the agents in order to improve performance. The algorithm defines what data has to be communicated and according to which rules areas can be reserved.

The rest of the paper is organized as follows: Section 2.2 explains the assumptions and principles of the basic algorithm, i.e., how areas are reserved and what kind of data needs to be communicated. Section 2.3 presents the implementation of the basic algorithm utilizing computational geometry. Section 2.4 covers an extension which employs waypoints to improve the performance. Agents are guided around congested areas instead of waiting until the situation changes. Section 2.5 shows results of the implementation in simulation and on a real system of autonomous robots, while Section 2.6 provides a conclusion and future prospects.

2.2 Description of the Basic Algorithm

This section is structured as follows: In the beginning the underlying assumptions and nomenclature are stated, followed by an introduction of reserved and requested areas, which govern the agents' motions. The algorithm depends on the agents having recent information about each other, which is indicated by a data flag. A handshaking procedure is employed to determine the state of the data flag. Possible conflicts between agents are resolved by a negotiation process, which assigns priority to certain agents. A presentation of the pseudo-code concludes the description of the basic algorithm.

The core of the algorithm is based on pair-wise conflict resolution. The basic concepts will be explained in terms of two agents i and j. The algorithm will then be expanded to n agents, mostly by treating every pair of agents separately. Cases where special rules apply if more than two agents are involved are explicitly mentioned. In general the algorithm is symmetric. Both agents operate according to exactly the same rules, with the only exception being how priority is assigned, see Section 2.2.6. For the

basic algorithm the following assumptions are made:

- Decentralized agents: all computation/control done on board
- Total number of agents n is known
- Motion primitives are available which move the agents in a deterministic fashion
- Point-to-point communication between agents
- Agents can localize themselves, but not others

Denote the global time with t. Every agent runs its own version of the algorithm in discrete time. Every discrete time step of the algorithm of an agent i is called a frame $k_i \in \mathbb{N}$. Note that uniform sampling times between time steps are not required. Frames are numbered in increasing order, s.t. $k_i(t_a) \geq k_i(t_b)$ if $t_a \geq t_b$.

Every agent i is trying to reach its task location $\mathbf{f}_i$. The selection of the task location is performed by some higher level entity (for example [31]) and outside the scope of this paper. Position and velocity of the agents are expressed in a global Cartesian coordinate system (x, y). For the purpose of the presented algorithm every agent i is modelled as a circular area $\mathbf{G}(\mathbf{p}_i)$ with radius r, the center being located at position $\mathbf{p}_i = (x_i, y_i)$. The current velocity of the agent is denoted by $\mathbf{v}_i = (\dot{x}_i, \dot{y}_i)$. The motion of i is controlled by a deterministic motion primitive MP(), which contains trajectory generation and low-level control of the actuators, for example [35]. The implementation of the motion primitive is beyond the scope of this paper. Upon specification of a desired destination $\mathbf{d}_i$ the motion primitive will compute a path that takes the agent to $\mathbf{d}_i$ with zero final velocity:

$$(\mathbf{p}_i(t), \mathbf{v}_i(t)) = \mathrm{MP}(\mathbf{p}_i(t_0), \mathbf{v}_i(t_0), \mathbf{d}_i(t_0), t), \quad t \geq t_0 \tag{2.1}$$

where $(\mathbf{p}_i(t), \mathbf{v}_i(t))$ denotes position and velocity of the trajectory, $\mathbf{p}_i(t_0)$ and $\mathbf{v}_i(t_0)$ are the position and velocity at the initial time t_0, and $\mathbf{d}_i(t_0)$ denotes the desired destination.

The dynamics of the agent are transparent to the path planner, although the planner can compute the entire trajectory ahead of time, given a triple $(\mathbf{p}_i, \mathbf{v}_i, \mathbf{d}_i)$. It is assumed that the vehicle will follow the precomputed trajectory of the motion primitive. Note that $\mathbf{d}_i$ and $\mathbf{f}_i$ do not have to be identical since the agent is allowed to stop or take a detour to avoid obstacles.

2.2.1 The data structure of the information state and the wireless communication

Every agent i stores information about itself and others in a data structure $T_{i,j,\text{data}}$, the information state. The first index $_i$ of T refers to the agent that owns the structure, the second index $_j$ denotes the agent that the information is about. The subsequent indices $_{\text{data}}$ denote the exact nature of the information. The content of T will be explained in more detail in the following sections.

The wireless communication is modelled as the exchange of data packets P between agents. The first index identifies the packet itself, while the subsequent indices stand for the particular type of information.

Table B.1 contains important variables as a reference. Bold indices indicate either areas or vectors, depending on the context, while normal indices indicate scalars.

2.2.2 The reserved area A and the requested area B

The basic principle of the algorithm is that every agent i has to be completely inside its reserved area $T_{i,i,\mathbf{A}}$ at all times. The algorithm ensures that the reserved areas of two different agents are never intersecting, which guarantees collision-avoidance in the

presence of arbitrary wireless delay. For a formal proof, see Appendix C.

The decentralized nature of the algorithm requires that agents go through a negotiation cycle in order to reserve areas. Instead of choosing $\mathbf{A}$ directly, agents are indicating their intentions by requesting areas $\mathbf{B}$ first, storing them in $T_{i,i,\mathbf{B}}$, and exchanging them with each other. Agents can change $\mathbf{B}$ arbitrarily. However, significant changes to $\mathbf{B}$ can cause the negotiation cycle to start over, see Section 2.2.3.

The requested area $T_{i,i,\mathbf{B}}$ has to contain the reserved area $T_{i,i,\mathbf{A}}$ at all times, $T_{i,i,\mathbf{A}}(t) \subseteq T_{i,i,\mathbf{B}}(t)$. Hence, an agent cannot possibly move to a location that is not inside the requested area. For performance reasons the final destination $\mathbf{f}_i$ should be included in $T_{i,i,\mathbf{B}}$. In general, $T_{i,i,\mathbf{B}}$ shrinks over time as i moves closer to its final destination, since it needs less room to maneuver.

Two intersecting areas $T_{i,i,\mathbf{B}}$ and $T_{j,j,\mathbf{B}}$ indicate a possible conflict. In this case the agents will negotiate to find out which one gets priority (Section 2.2.6) and how the areas $\mathbf{A}$ are being selected (Section 2.2.7). The basis for this negotiation is a scalar cost function.

Figure 2.1 visualizes the process of requesting and reserving areas. Initially (upper left) both i and j have distinct requested areas, and no collisions are possible. Upon changing $\mathbf{f}_i$, i indicates its intentions by sending a new $T_{i,i,\mathbf{B}}$ to j (upper right). The agents exchange their respective costs. Agent i is being assigned priority over j, which withdraws by reducing the size of $T_{j,j,\mathbf{A}}$ (lower left). Once i receives a message from j confirming the change of $T_{j,j,\mathbf{A}}$, it will reserve a new area and move on towards $\mathbf{f}_i$ (lower right).

Note that this process runs while the agents are in motion, guaranteeing safety at any time. Agents will recompute costs and renegotiate paths en route, which allows them

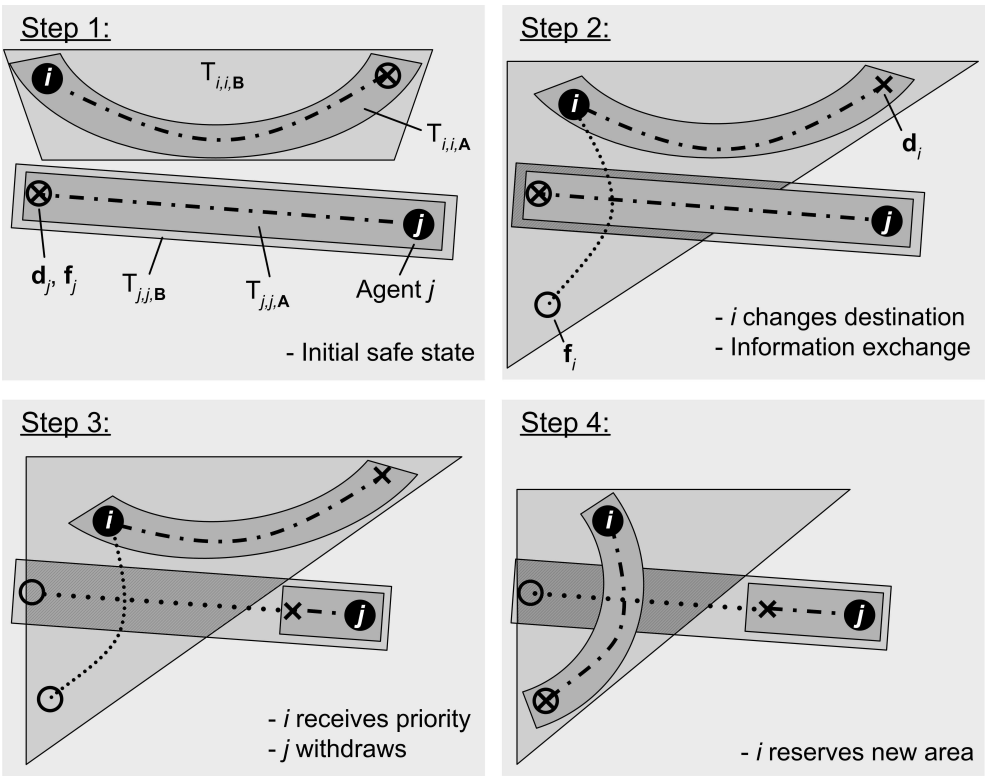

Figure 2.1: Negotiation process

to quickly respond to changing situations. In general, a more responsive system also requires more communication between agents.

2.2.3 The data flag D and the handshake

The data flag $T_{i,j,D}$ captures the condition of i's information state with respect to j: it is set high ($T_{i,j,D} = 1$) if i has current information about j, and is set low ($T_{i,j,D} = 0$) otherwise (default). In order to guarantee safety, i can only change $T_{i,i,\mathbf{A}}$ if $T_{i,j,D} = 1$, i.e., if it knows that it will not interfere with any possible motion of j.

The data flag $T_{i,j,D}$ is set low at time $t_{D0} = t(k_i)$ if agent i requests an area $T_{i,i,\mathbf{B}}(k_i)$ such that $T_{i,i,\mathbf{B}}(k_i) \not\subseteq T_{i,i,\mathbf{B}}(k_i - 1)$ or if i receives a message from j at k_i, indicating that

$T_{i,j,\mathbf{B}}(k_i) \not\subseteq T_{i,j,\mathbf{B}}(k_i - 1)$.

In order to set the data flag $T_{i,j,D}$ high the agents i and j have to go through a handshaking procedure, which ensures that:

- Agent i has knowledge of the current $T_{j,j,\mathbf{B}}$
- Agent i will only accept information from j that is more recent than t_{D0}
- Packets that are received out of order are discarded (i.e. P_{q1} is received after P_{q2}, although P_{q1} was sent before P_{q2})

Figure 2.2 shows the information flow during a handshake starting with i. The horizontal axis denotes the global time t. Solid arrows represent packets sent between the agents. Beginning and end points of the arrows denote the times of sending and receiving a packet. Characteristic times of the process are indicated with $t_\bullet$ on the global time axis:

- $t_{i,1}$: i sends a packet to j, requesting recent information
- $t_{i,2}$: j sends the requested update to i
- $t_{i,3}$: i receives the update from j, setting $T_{i,j,D} = 1$

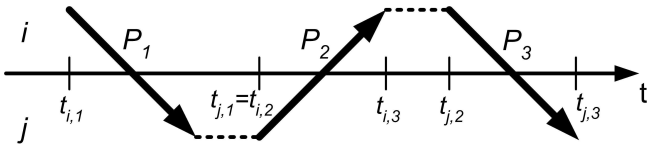

Figure 2.2: Data flow during a handshake

Agent i initializes the handshake by sending recent data P_1 to j while at the same time requesting current information. As soon as i receives the return message P_2 from j, i stores the data which completes the handshake. Agent i now has guaranteed recent

information about j and sets $T_{i,j,D} = 1$. In most cases, the agents have to go through a "double-handshake", since the procedure has to be executed with respect to i and j. In practice that requires only one additional message P_3 to be sent from i to j.

Formally, the handshake between i and j is accomplished by setting time stamps $T_{i,j,t1}$, $T_{i,j,t2}$, $T_{j,i,t1}$, and $T_{j,i,t2}$, which are stored internally within the agents. This resolves the differences in the local clocks of the agents, k_i and k_j. Table 2.1 describes how these time stamps are set.

Table 2.1: Handshaking procedure, starting with agent i

Time t	Action	Example
$t_{D0} = t_{i,1}$	**Agent** i: reset information state:	$t = 11$, $k_i = 115$
	$T_{i,j,D} = 0$, $T_{i,j,t1} = -1$, $T_{i,j,t2} = -2$	
	$T_{i,j,t1} = k_i(t_{i,1})$, $P_{1,t,\text{me}} = T_{i,j,t1}$	$T_{i,j,t1} = P_{1,t,\text{me}} = 115$
	$P_{1,t,\text{other}} = T_{i,j,t2}$, send P_1	$P_{1,t,\text{other}} = -2$
$t_{j,1} = t_{i,2}$	**Agent** j: receive P_1, $T_{j,i,t2} = P_{1,t,\text{me}}$	$t = 23$, $k_j = 216$, $T_{j,i,t2} = 115$
	reset information state:	
	$T_{j,i,D} = 0$, $T_{j,i,t1} = -1$	
	$T_{j,i,t1} = k_j(t_{j,1})$, $P_{2,t,\text{me}} = T_{j,i,t1}$	$T_{j,i,t1} = P_{2,t,\text{me}} = 216$
	$P_{2,t,\text{other}} = T_{j,i,t2}$, send P_2	$P_{2,t,\text{other}} = 115$
$t_{i,3}$	**Agent** i: receive P_2, $T_{i,j,t2} = P_{2,t,\text{me}}$	$t = 28$, $T_{i,j,t2} = 216$
	$T_{i,j,D} = 1$	
$t_{j,2}$	**Agent** i: $P_{3,t,\text{me}} = k_i(t_{j,2})$	$t = 37$, $k_i = 144$
	$P_{3,t,\text{other}} = T_{i,j,t2}$, send P_3	$P_{3,t,\text{me}} = 144$, $P_{3,t,\text{other}} = 216$
$t_{j,3}$	**Agent** j: receive P_3, $T_{j,i,t2} = P_{3,t,\text{me}}$	$t = 43$, $T_{j,i,t2} = 144$
	$T_{j,i,D} = 1$	

The variable $T_{j,i,t2}$ denotes the frame k_i when i sent the last packet to j. Every packet (for example P_1) contains a time stamp $P_{1,t,\text{me}}$, which is the frame k_i of the sender i when P_1 was sent. A packet is only accepted by the receiver j if $P_{1,t,\text{me}} > T_{j,i,t2}$. If a packet is accepted, the information state of j is updated, $T_{j,i,t2} = P_{1,t,\text{me}}$. Since k_i is strictly increasing in time $P_{1,t,\text{me}} \leq T_{j,i,t2}$ indicates a packet which has been received out of order, i.e., j already received a packet P_q which was sent after P_1 was sent. In that case P_1 is discarded.

The time stamp $T_{i,j,t1}$ denotes the frame $k_i(t_{i,1})$ when i sent the first packet P_1 to j after t_{D0}. The variable $T_{i,j,t1}$ is used to determine if packets from j contain information that is more recent than t_{D0}. Every packet from j to i (for example P_2) contains a time stamp $P_{2,t,\text{other}} = T_{j,i,t2}$, which effectively returns the time stamp of the last packet sent from i to j, $P_{1,t,\text{me}}$. By comparing $P_{2,t,\text{other}}$ (the returned time stamp) to $T_{i,j,t1}$ (i's stored time stamp) it is possible to determine if P_2 was sent after t_{D0}, since both time stamps are expressed in i's time frame. If $P_{2,t,\text{other}} \geq T_{i,j,t1}$ then i has a guarantee that P_2 was sent after t_{D0}, otherwise P_2 is discarded.

Table 2.1 contains an example of how the handshake is conducted. During the initialization i sets the time stamp $T_{i,j,t1} = 115$ which is sent to j, stored in $T_{j,i,t2}$, and is reported back to i as $P_{2,t,\text{other}}$. Agent i finalizes its handshake upon receiving P_2. At any time after $t = 11$, i will only accept messages P_q with $P_{q,t,\text{other}} \geq 115$, since that indicates that the sender of P_q received P_1. Agent j's handshake is initialized at $t = 23$ by setting $T_{j,i,t1} = 216$, which is sent to i as $P_{2,t,\text{me}}$ and reported back to j as $P_{3,t,\text{other}}$.

2.2.4 Information updates

Every agent can send information to any other agent at any time. These information updates serve to increase the performance by allowing the agents to respond to the cur-

rent situation. In practice there has to be a trade-off between performance and wireless traffic. There are situations where an information update yields a significant performance increase, while at other times the performance gain is marginal. The triggering events when to send an information update serve as a tuning parameter, depending on how much wireless bandwidth is available.

A packet P_q from i to j is accepted as an update if $P_{q,\text{me}} > T_{j,i,t2}$ and $P_{q,\text{other}} \geq T_{j,i,t1}$. A successful update does not alter $T_{j,i,t1}$, but it changes $T_{j,i,t2} = P_{q,t,\text{me}}$, so that further messages from i only get accepted if they are sent after the new $T_{j,i,t2}$. Note that the final packet of a handshake is a regular information update.

2.2.5 The communication flag F

The purpose of the communication flag $T_{i,j,F}$ is to reduce the amount of wireless traffic. It is set high if i has to communicate with j (the default), low otherwise. Once the data flag $T_{i,j,D}$ is set high i compares $T_{i,i,\mathbf{B}}$ and $T_{i,j,\mathbf{B}}$. If these two areas do not intersect there is no possible conflict, and wireless communication between agents i and j is no longer required. Therefore, $T_{i,j,F}$ will be set low. As soon as i resets $T_{i,j,D} = 0$, $T_{i,j,F}$ will be set high again.

2.2.6 The priority flag R

In case of a possible conflict reserved areas are assigned based on the notion of priority. The priority flag $T_{i,j,R}$ is set high if i has priority over j. The agent with priority may select a reserved area that intersects the requested area of the other agent, which is not permitted in general. The status of the flag R is determined based on a scalar cost function, which is computed by every agent for itself and then sent to the other agent.

The agent with the lower cost gets priority. Note that the choice of the cost function affects performance but not safety.

The procedure how priority is assigned guarantees that $T_{i,j,R}(t) = T_{j,i,R}(t) = 1$ is not possible at any time t. One agent, labelled the priority assigner (PA), is making an active decision about who should have priority. The other agent (PR) is accepting the decision of the PA. The selection of which agent is the PA is arbitrary, but has to be fixed. Note that this is the only aspect of the algorithm where agents are treated differently from each other. The PA receives the cost from the PR, makes the decision who should have priority, and sends the result back to the PR, which changes R accordingly. The PA makes sure that during the times when R is transferred from one agent to the other, neither of the two has priority. The priority flag R is assigned or revoked by regular information updates.

Before the PA (denoted by i without loss of generality) can assume priority itself, it has to ensure that the PR (denoted by j) has given up priority. For that reason i sends a packet to j requiring it to set $T_{j,i,R} = 0$. As soon as i receives a return message confirming that $T_{j,i,R} = 0$, i can set $T_{i,j,R} = 1$. In order to prevent $T_{i,j,R}(t) = T_{j,i,R}(t) = 1$ in the presence of significantly delayed messages, i sets a time stamp $T_{i,j,t5} = k_i(t)$ when it requires j to give up R. It will only accept an answer from j if the message is more recent than $T_{i,j,t5}$.

2.2.7 Changing the reserved area A

Every agent i can change $T_{i,i,\mathbf{A}}$ at any given time k_i according to

$$T_{i,i,\mathbf{A}}(k_i + 1) \subseteq (T_{i,i,\mathbf{A}}(k_i) \cup [T_{i,i,\mathbf{B}}(k_i) \setminus T_{i,j,\mathbf{A}}(k_i)]), \quad \text{if } [(T_{i,j,D}(k_i) = 1) \wedge (T_{i,j,R}(k_i) = 1)] \tag{2.2}$$

$$T_{i,i,\mathbf{A}}(k_i+1) \subseteq (T_{i,i,\mathbf{A}}(k_i) \cup [T_{i,i,\mathbf{B}}(k_i) \setminus T_{i,j,\mathbf{B}}(k_i)]), \quad \text{if } [(T_{i,j,D}(k_i)=1) \wedge (T_{i,j,R}(k_i)=0)] \tag{2.3}$$

$$T_{i,i,\mathbf{A}}(k_i+1) \subseteq T_{i,i,\mathbf{A}}(k_i), \quad \text{otherwise} \tag{2.4}$$

as long as there exists a temporary destination $\mathbf{d}_{i,\text{tp}}(k_i+1)$, such that

$$\mathbf{G}(\mathbf{p}_{i,\text{tp}}(t)) \subseteq T_{i,i,\mathbf{A}}(k_i+1), \quad \forall t \geq t_0 \tag{2.5}$$

$$(\mathbf{p}_{i,\text{tp}}(t), \mathbf{v}_{i,\text{tp}}(t)) = \text{MP}(\mathbf{p}_i(t_0), \mathbf{v}_i(t_0), \mathbf{d}_{i,\text{tp}}(k_i+1), t) \tag{2.6}$$

where t_0 is the time when the reserved area is changed to $T_{i,i,\mathbf{A}}(k_i+1)$. Note that there is no requirement for i to keep the temporary destination at $\mathbf{d}_{i,\text{tp}}(k_i+1)$, as long as it is guaranteed that i is always fully contained by $T_{i,i,\mathbf{A}}(k_i+1)$ while moving.

The rules (2.2) through (2.4) guarantee that for all t, $T_{i,i,\mathbf{A}}(t) \cap T_{j,j,\mathbf{A}}(t) = \emptyset$, since i can only change $T_{i,i,\mathbf{A}}$ if it has recent information about j (high data flag $T_{i,j,D}$). It will always stay inside $T_{i,i,\mathbf{B}}$ and either avoid $T_{i,j,\mathbf{A}}$ or $T_{i,j,\mathbf{B}}$, depending on whether it has priority. If no change is possible (either if $T_{i,j,D} = 0$ or no different path can be found) i can always maintain the previous $T_{i,i,\mathbf{A}}(k_i)$. The distinction of the reserved areas of i and j guarantees safety. As far as performance is concerned, the reserved area should be chosen such that it includes $\mathbf{f}_i$.

2.2.8 Pseudo-code of the algorithm

This Section contains the pseudo-code of the main loop for agent i. This loop is run once every frame k_i. The function MAX() returns the maximum argument. In the following, line l of the pseudo-code is being referred to as pc:l.

1. Initialize: determine $\mathbf{p}_i$, $\mathbf{v}_i$, set $T_{i,j,\text{send}} = 0$

2. Receive data: receive data packet P_{in} from j

3. Update information state:

   (a) IF ($P_{\text{in},t,\text{me}} > T_{i,j,t2}$)

   (b) $T_{i,j,t2} = P_{\text{in},t,\text{me}}$

   (c) IF ($P_{\text{in},reqD} == 1$) $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 1)$

   (d) IF [($T_{i,j,D} == 1$) & ($P_{\text{in},\mathbf{B}} \not\subseteq T_{i,j,\mathbf{B}}$)]

   (e) $T_{i,j,D} = 0$, $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 2)$, $T_{i,j,F} = 1$, $T_{i,j,t1} = -1$

   (f) IF ($P_{\text{in},t,\text{other}} \geq T_{i,j,t1}$)

   (g) $T_{i,j,D} = 1$, $T_{i,j,\mathbf{A}} = P_{\text{in},\mathbf{A}}$, $T_{i,j,\mathbf{B}} = P_{\text{in},\mathbf{B}}$, $T_{i,j,c,\text{other}} = P_{\text{in},c}$

   (h) IF (isPA)

   (i) IF [($T_{i,j,R,\text{other}} = 0$) & ($P_{\text{in},R} = 0$) & ($P_{\text{in},t,\text{other}} \geq T_{i,j,t5}$)]

   (j) $T_{i,j,R} = 1$

   (k) ELSE

   (l) IF ($P_{\text{in},R,\text{other}} == 0$) $T_{i,j,R} = 0$

   (m) ELSE IF ($P_{\text{in},R,\text{other}} == 1$) $T_{i,j,R} = 1$

4. Change final destination: If reached $\mathbf{f}_i$, set new $\mathbf{f}_i$

5. Update requested area:

   (a) Determine new requested area $T_{i,i,\mathbf{B}}(k_i + 1)$

   (b) IF ($T_{i,i,\mathbf{B}}(k_i + 1) \not\subseteq T_{i,i,\mathbf{B}}(k_i)$)

   (c) $T_{i,j,D} = 0$, $T_{i,j,R} = 0$, $T_{i,j,t1} = -1$, $T_{i,j,t2} = -2$, $T_{i,j,F} = 1$,

   (d) $T_{i,j,R,\text{other}} = -1$, $T_{i,j,t5} = -1$, $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 2)$

6. Update communication flag:

(a) IF [($T_{i,j,D}$ == 1) & ($T_{i,i,\mathbf{B}} \cap T_{i,j,\mathbf{B}} = \emptyset$) & ($T_{i,j,F}$ == 1)]

(b) $T_{i,j,F} = 0$, $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 1)$

7. Decide priority:

(a) Compute cost $T_{i,j,c}$

(b) IF [($T_{i,j,D}$ == 1) & (isPA)]

(c) IF ($T_{i,j,c} \leq T_{i,j,c,\text{other}}$)

(d) IF ($T_{i,j,R,\text{other}} \neq 0$)

(e) $T_{i,j,R,\text{other}} = 0$, $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 2)$, $T_{i,j,t5} = k_i + 1$

(f) ELSE

(g) $T_{i,j,R} = 0$

(h) IF ($T_{i,j,R,\text{other}} \neq 1$) $T_{i,j,R,\text{other}} = 1$, $T_{i,j,\text{send}} = \text{MAX}(T_{i,j,\text{send}}, 1)$

8. Change reserved area $T_{i,i,\mathbf{A}}(k_i + 1)$

9. Send data:

(a) IF ($T_{i,j,\text{send}} > 0$)

(b) IF ($T_{i,j,t1} == -1$) $T_{i,j,t1} = k_i + 1$

(c) $P_{\text{out},\mathbf{A}} = T_{i,i,\mathbf{A}}(k_i + 1)$, $P_{\text{out},\mathbf{B}} = T_{i,i,\mathbf{B}}(k_i + 1)$, $P_{\text{out},t,\text{me}} = k_i + 1$

(d) $P_{\text{out},t,\text{other}} = T_{i,j,t2}$, $P_{\text{out},R} = T_{i,j,R}$, $P_{\text{out},R,\text{other}} = T_{i,j,R,\text{other}}$

(e) IF [($T_{i,j,\text{send}} == 2$) | ($T_{i,j,D} == 0$)] $P_{\text{out},reqD} = 1$

(f) Send packet P_{out} to j

10. Move agent

Initialize

The current position and velocity are determined by estimation routines which are outside the scope of this paper, for an example see [37].

Receive data

Agent i retrieves the data packets that have been buffered since the last execution of the main loop. It is assumed that the wireless system can receive and buffer data independently of the main loop.

Update information state

The information state T_i is updated by P_{in}. The packet is only accepted if it contains recent data (pc:3a and pc:3f), see Section 2.2.4. The flag $P_{\text{in},reqD}$ is set high if j requests a return message. The flag $T_{i,j,\text{send}}$ keeps track of whether i has to send a packet to j. At the beginning of the main loop $T_{i,j,\text{send}}$ is set to zero. If $T_{i,j,\text{send}} > 0$, i has to send a message to j, while $T_{i,j,\text{send}} = 2$ indicates that i is requesting an answer from j in addition. Line pc:3d checks if a new handshake has to be initiated. Initialization is accomplished by setting the data flag low. If the packet contains valid data, the information state T_i is updated by copying **A**, **B**, and the cost. The remaining lines handle the priority flag. If i is the PA then it can only assume priority if j gave up priority first by setting $P_{\text{in},R} = 0$ and using a recent time stamp $P_{\text{in},t,\text{other}} \geq T_{i,j,t5}$. If i is not the PA it follows the directives stored in $P_{\text{in},R,\text{other}}$.

Change final destination

The destination is considered to be reached if the agent arrives at $\mathbf{f}_i$ with zero velocity, given some tolerance. In that case, the agent selects a new destination following some higher-level directive.

Update requested area

The requested area is recomputed as a function of $\mathbf{p}_i(k_i)$, $\mathbf{v}_i(k_i)$, $\mathbf{d}_i(k_i)$, $\mathbf{f}_i(k_i)$, and $T_{i,i,\mathbf{A}}(k_i)$, see Section 2.3.3. Depending on $T_{i,i,\mathbf{B}}(k_i + 1)$ a new handshake might have to be initialized by resetting T_i, see Section 2.2.3.

Update communication flag

In case the communication flag is set low (see Section 2.2.5) i informs j, so that j can set $T_{j,i,F} = 0$ as well.

Decide priority

The first step is to compute the cost $T_{i,j,c}$ according to the chosen cost function. If agent i is the PA it decides who should have priority by comparing its own cost to that reported by j. The flag $T_{i,j,R,\text{other}}$ serves two purposes: it stores the last decision in order to determine when a change in priority occurred, and it contains the desired priority state of j. If the PA decides to have priority, it sets $T_{i,j,t5}$ and sends a message to j, requesting a confirming return message, see Section 2.2.6.

Change reserved area

Agent i changes the reserved area $T_{i,i,\mathbf{A}}$ according to the rules in Section 2.2.7.

Send data

If the handshake with j has been initialized ($T_{i,j,t1} = 0$) in this frame then $T_{i,j,t1}$ is set to the frame number $k_i + 1$. The packet is assembled by copying the required data. In case either $T_{i,j,\text{send}} = 2$ or a new handshake is initialized ($T_{i,j,D} = 0$) the $P_{\text{out},reqD}$ flag is set high, requesting an information update from j.

Move agent

The actual motion of the agent is determined by the temporary destination $\mathbf{d}_i(k_i + 1)$, which is an input to the motion primitive. The agent can only select a destination $\mathbf{d}_i$ if the resulting path is completely contained within $T_{i,i,\mathbf{A}}(k_i + 1)$, see Section 2.2.7. The rules for selecting $T_{i,i,\mathbf{A}}$ guarantee that such a destination exists. For performance reasons $\mathbf{d}_i$ should be chosen as close to $\mathbf{f}_i$ as possible.

2.2.9 Extension to more than two agents

If more than two agents are involved the algorithm is extended in a straightforward manner by applying the operations to all agents $j \neq i$. Every agent can receive packets from any other agent every frame. All received packets are stored in a queue and used to update the information state. The order in which they are processed is irrelevant since the time stamps ensure that only the most recent information is stored in T_i. The information state has to be reset for all $j \neq i$ if a significant change in $T_{i,i,\mathbf{B}}$ occurred

(pc:5b). The communication flag F has to be updated for all $j \neq i$ (pc:6a). Priority decisions are made for any $j \neq i$ where i is the PA with respect to j. Out of every pair of agents one agent can have priority, not both. The change of $T_{i,i,\mathbf{A}}$ has to respect the data and priority flags of all other agents. For example, if there exists $j \neq i$, s.t. $T_{i,j,D}(k_i) = 0$, then the only choice left for i is (2.4). It follows that i has to perform a handshake with all $j \neq i$ before it can reserve an area that is not a subset of the previously reserved area. This is the only part of the algorithm that requires i to contact every other agent. Packets are being sent to all agents that require an information update. Example:

> Assume there are three agents: a_1, a_2, and a_3. Further assume that initially the reserved areas of all three agents do no intersect. If a_1 decides to change its requested area $T_{1,1,\mathbf{B}}$ it has to initiate a handshake with a_2 and a_3. This means that the data flags $T_{1,2,D}$ and $T_{1,3,D}$ are set low and a_1 has to keep its current reserved area $T_{1,1,\mathbf{A}}$ or a subset thereof, which is safe by the initial assumption. Once the handshakes with both a_2 and a_3 are complete ($T_{1,2,D} = T_{1,3,D} = 1$) agent a_1 can choose a different reserved area according to the rules (2.2) through (2.4), where the rules have to hold for both j=a_2 and j=a_3.

2.3 Implementation of the Basic Algorithm

This section presents an implementation of the algorithm by utilizing computational geometry. Real-time application demands the geometrical shapes to be kept simple while not compromising performance. Polygons are selected to represent the areas **A** and **B**, since they can approximate an arbitrary shape while being computationally attractive. The number of polygon vertices is a tuning parameter that can be chosen depending on the computational power and wireless bandwidth available. The selection

process of **A** and **B** is driven by the choice of temporary and final destinations, $\mathbf{d}_i$ and $\mathbf{f}_i$.

2.3.1 Trajectory approximation

It is desirable to include the dynamics of the vehicle when computing **A** and **B** in order to keep the areas comparatively small, which in turn reduces the interaction between agents and increases performance. The motion primitive (2.1) captures the agent dynamics by computing a dynamically feasible path, which is then approximated by a set of rectangles, see Figure 2.3.

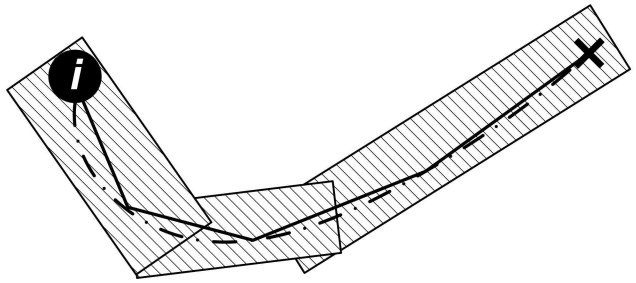

Figure 2.3: Approximation of trajectory

The first step is to fit line segments (represented by full lines) to the path (dash-dotted curve) in a recursive procedure. The endpoints of the initial line segments are identical to the endpoints of the path. If the maximum deviation between the line segment and the curve is larger than a specified threshold, the line segment is split into two. The new endpoint is the point on the curve with the maximum deviation. This procedure is repeated until the maximum error is smaller than the threshold.

Finally, rectangles are fitted around the line segments in order to account for the non-zero radius of the agent. The maximum width of the rectangles can be specified. The wider the rectangles the fewer rectangles are needed to approximate the path, but

the poorer the approximation. An additional buffer width is added to the size of the rectangles in order to account for process noise during the motion of the agent. The final trajectory approximation is a set of rectangles.

2.3.2 Computation of B

The requested area has to include the current reserved area and should include areas that are possibly required to reach the final destination $\mathbf{f}_i$. On the one hand the area should be large enough to allow some maneuvering given the agent's dynamics. On the other hand, the smaller the area **B** the less interaction with other agents and the less wireless traffic.

The area **B** is found by computing the convex hull of the set of points Q that have to be included in **B**. The convexity of the resulting area allows an efficient execution of some geometrical operations which are required by the algorithm. Figure 2.4 depicts the construction of the requested area.

The vertices of $T_{i,i,\mathbf{A}}$ make up the the first points of Q. In order to give i room to maneuver, the triangle formed by the points $\mathbf{p}_i$, $\mathbf{d}_i$, and $\mathbf{f}_i$ is considered. Since the agent has non-zero radius, the triangle is replaced by a polygon $\mathbf{B}_{\text{tri}}$, s.t. the agent fits exactly into each corner. The six vertices of the polygon are added to Q. The motion of i is characterized by two corner cases: i moves all the way to $\mathbf{d}_i$ and i moves directly to $\mathbf{f}_i$. These two cases are included by computing the trajectories and approximating them by sets of rectangles $\mathbf{B}_{p,d}$ and $\mathbf{B}_{p,f}$, see Section 2.3.1. The vertices of $\mathbf{B}_{p,d}$ and $\mathbf{B}_{p,f}$ are added to Q.

Once the entire set of points Q is determined, the convex hull is computed using a Graham Scan, see [19].

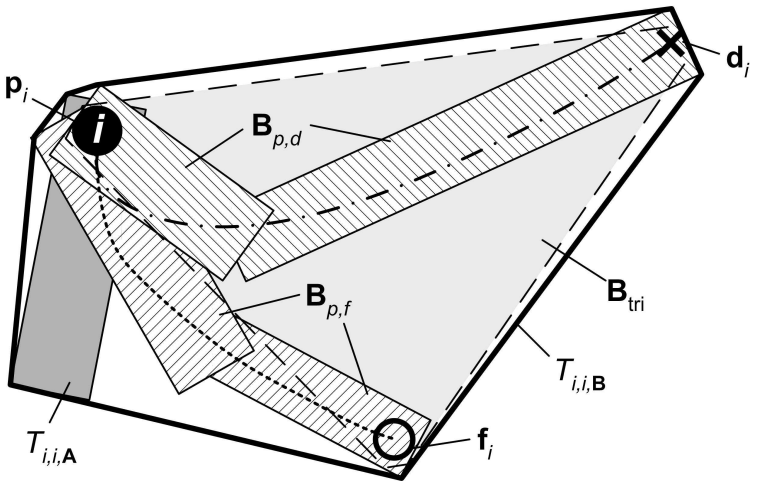

Figure 2.4: Computation of **B**

2.3.3 Update of B

On line pc:5a a new requested area is determined depending on the current situation. In general, the agent needs less room to maneuver as it closes in on the destination and so the area shrinks over time. This increases the performance of the system by reducing the interaction between agents. If the final destination $\mathbf{f}_i$ has not changed in this particular frame, a candidate area $\mathbf{B}_{\text{cand}}$ is being computed using the procedure given in Section 2.3.2. The area $\mathbf{B}_{\text{cand}}$ is accepted as $T_{i,i,\mathbf{B}}(k_i + 1)$ if it is a subset of $T_{i,i,\mathbf{B}}(k_i)$, otherwise $T_{i,i,\mathbf{B}}(k_i + 1) = T_{i,i,\mathbf{B}}(k_i)$.

2.3.4 Selection of d and A

In the proposed implementation of the algorithm the selection of $T_{i,i,\mathbf{A}}$ is firmly linked with $\mathbf{d}_i$. On line pc:8 $T_{i,i,\mathbf{A}}$ is set to the rectangle approximation of the path from $\mathbf{p}_i$ to $\mathbf{d}_i$ as described in Section 2.3.1. This guarantees that i's path is always fully contained in

$T_{i,i,\mathbf{A}}$, but it puts additional constraints on $\mathbf{d}_i$, since $\mathbf{d}_i$ can now only be chosen such that the resulting $T_{i,i,\mathbf{A}}$ is valid, i.e., obeys the rules (2.2) through (2.4).

The temporary destination $\mathbf{d}_i$ is selected by performing a search on the line segment between $\mathbf{p}_i$ and $\mathbf{f}_i$. The objective is to find the point closest to $\mathbf{f}_i$ which results in a valid $T_{i,i,\mathbf{A}}$. Additionally, points are discarded which yield a $T_{i,i,\mathbf{A}}$ that intersects any $T_{i,j,\mathbf{B}}$ with $T_{i,j,R} = 0$. This is done to allow agents with priority to pass, even though it is not required by the safety rules.

2.3.5 Definition of the cost function

The scalar cost function is defined for agent i with respect to all other agents j. In the presented implementation the cost function is a heuristic with the objective to minimize the waiting time of the agents. Define controversial areas as the intersections between $T_{i,i,\mathbf{B}}$ and any $T_{i,j,\mathbf{B}}$, $j \neq i$. The underlying idea is that an agent who is close to a controversial area is given priority over an agent who is farther away, since the first agent might resolve the conflict before the other agent even comes close. The basic cost is therefore the distance from p_i to $T_{i,j,\mathbf{B}}$. This basic cost is augmented by requested and reserved areas on i's path, since these are likely to affect the motion of i.

The cost function is computed by identifying three types of events

- **Enter**($T_{i,h,\mathbf{A}}$): the location on i's path where the path intersects $T_{i,h,\mathbf{A}}$ for the first time
- **Enter**($T_{i,h,\mathbf{B}}$): the location on i's path where the path intersects $T_{i,h,\mathbf{B}}$ for the first time
- **Leave**($T_{i,h,\mathbf{B}}$): the location on i's path where the path intersects $T_{i,h,\mathbf{B}}$ for the last time

where $h \neq i, h \neq j$. These events are determined by computing the intersections between the line segments which approximate i's path and the respective areas $T_{i,h,\mathbf{A}}$ or $T_{i,h,\mathbf{B}}$. The first intersection point along the path is then denoted by $\mathbf{Enter}(T_{i,h,\bullet})$ and the last intersection point by $\mathbf{Leave}(T_{i,h,\bullet})$. The cost of i with respect to j is defined as

$$T_{i,j,c} \quad = \quad \text{dist}(\mathbf{p}_i, \mathbf{Leave}(T_{i,j,\mathbf{B}})) + c_{\text{pen},A} b_{\text{enter},A} + c_{\text{pen},B} b_{\text{enter},B} \tag{2.7}$$

where $\text{dist}(\mathbf{p}_1, \mathbf{p}_2)$ denotes the distance from $\mathbf{p}_1$ to $\mathbf{p}_2$, measured along the path. The constant $c_{\text{pen},A}$ ($c_{\text{pen},B}$) denotes the penalty cost for crossing a reserved (requested) area. The variable $b_{\text{enter},A}$ ($b_{\text{enter},B}$) denotes how many $\mathbf{Enter}(T_{i,h,\mathbf{A}})$ ($\mathbf{Enter}(T_{i,h,\mathbf{B}})$) events are encountered on the path between $\mathbf{p}_i$ and $\mathbf{Leave}(T_{i,j,\mathbf{B}})$, $h \neq j$. The penalty costs $c_{\text{pen},A}$ and $c_{\text{pen},B}$ are part of the heuristic and act as tuning parameters. Figure 2.5 visualizes the computation of the cost function.

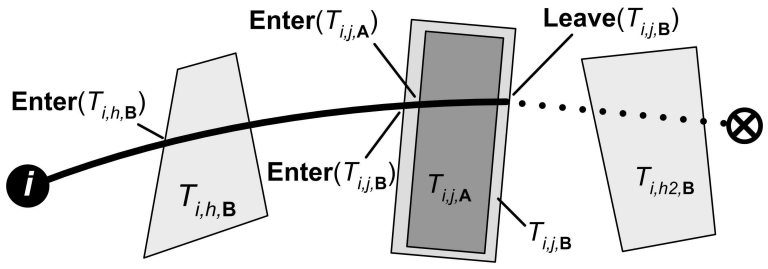

Figure 2.5: Example of cost function

The full curve denotes the basic distance $\text{dist}(\mathbf{p}_i, \mathbf{Leave}(T_{i,j,\mathbf{B}}))$, while the dotted curve denotes the part of the path that does not affect $T_{i,j,c}$. For the example given in Figure 2.5 the final cost is equal to

$$T_{i,j,c} \quad = \quad \text{dist}(\mathbf{p}_i, \mathbf{Leave}(T_{i,j,\mathbf{B}})) + c_{\text{pen},A} + c_{\text{pen},B} \tag{2.8}$$

since the path enters one requested area ($T_{i,h,\mathbf{B}}$) and one reserved area ($T_{i,j,\mathbf{A}}$) before it reaches $\mathbf{Leave}(T_{i,j,\mathbf{B}})$. The event $\mathbf{Enter}(T_{i,h2,\mathbf{B}})$ incurs no penalty, since it is encountered

after **Leave**($T_{i,j,\mathbf{B}}$). Entering $T_{i,j,\mathbf{B}}$ is never penalized, since this event has to occur in any case.

2.3.6 Required geometric operations

Two recurring geometric operations are required to run the algorithm: *isABinB*($\mathbf{N}_1, \mathbf{N}_2$), which checks whether area $\mathbf{N}_1$ is a subset of area $\mathbf{N}_2$, and *isABintersectingAB*($\mathbf{N}_1, \mathbf{N}_2$), which computes whether areas $\mathbf{N}_1$ and $\mathbf{N}_2$ intersect.

The function *isABinB*($\mathbf{N}_1, \mathbf{N}_2$) is utilized on line pc:5b and on line pc:8 when computing a new $T_{i,i,\mathbf{A}}$. Note that $\mathbf{N}_2$ is always a reserved area and therefore convex. The implementation uses a point-in-polygon algorithm [36], which checks whether all vertices of $\mathbf{N}_1$ are located inside $\mathbf{N}_2$. If that is the case, then $\mathbf{N}_1 \subseteq \mathbf{N}_2$ due to the convexity of $\mathbf{N}_2$.

Line pc:8 utilizes *isABintersectingAB*($\mathbf{N}_1, \mathbf{N}_2$) in order to determine whether $T_{i,i,\mathbf{A}}$ intersects any reserved or requested areas. This is accomplished by computing whether the line segments of $\mathbf{N}_1$ are intersecting any line segments of $\mathbf{N}_2$ [19]. In case no intersections occur, it has to be determined if one area is completely contained by the other. This check is performed by using the aforementioned point-in-polygon check in order to determine if a random vertex of $\mathbf{N}_1$ is located inside $\mathbf{N}_2$ and vice versa.

Note that all of these algorithms were selected with ease of implementation in mind and run in real-time.

2.3.7 Computational complexity

The computational complexity of the presented algorithm is a function of the number of involved agents n and the maximum number of line segments m required to represent the areas **A** and **B**.

Most operations of the algorithm are either of order $O(1)$ if they have to be performed once every frame (eg. pc:1, pc:4) or of order $O(n)$ if they have to be performed for every agent (e.g. pc:2). The only exceptions are the computation of $T_{i,i,\mathbf{B}}$ on pc:5a, the check for $T_{i,i,\mathbf{B}}(k_i+1) \not\subseteq T_{i,i,\mathbf{B}}(k_i)$ on pc:5b, the computation of the cost on pc:7a, and the change of $T_{i,i,\mathbf{A}}$ on pc:8.

In the current implementation, the computation of $T_{i,i,\mathbf{A}}$ involves computing a path ($O(1)$) using the motion primitive (2.1), breaking it up into $O(m)$ line segments and fitting rectangles around them. The process of fitting the rectangles is $O(1)$ for every line segment, hence the entire process is of order $O(m)$.

When computing $T_{i,i,\mathbf{B}}$ an agent has to manipulate $O(m)$ vertices: $O(m)$ vertices which describe the current $T_{i,i,\mathbf{A}}$ and $O(m)$ vertices from the two newly linearized trajectories. The computation of the convex hull is a $O(w \log w)$ operation, where w is the number of vertices involved. The process of computing $T_{i,i,\mathbf{B}}$ is therefore an $O(m \log m)$ operation.

The check whether $T_{i,i,\mathbf{B}}(k_i + 1) \not\subseteq T_{i,i,\mathbf{B}}(k_i)$ is a $O(m^2)$ operation. Ensuring that a point is located inside a polygon with m vertices has complexity $O(m)$. This calculation has to be performed for all $O(m)$ vertices of $T_{i,i,\mathbf{B}}(k_i + 1)$.

The cost $T_{i,j,c}$ has to be computed for i with respect to all other agents j. The first step is to compute a new area $\mathbf{A}_{\text{new}}$ ($O(m)$). Computing all events is $O(nm^2)$, since the

check whether a line segment intersects a polygon $\mathbf{A}$ or $\mathbf{B}$ is $O(m)$ and this operation has to be performed nm times, once for every line segment in $\mathbf{A}_{\text{new}}$ and every agent j. Every agent j can be associated with up to three events, hence the total number of events is $O(n)$. The events are then sorted ($O(n \log n)$) and evaluated ($O(n^2)$). The dominant complexity terms are $O(nm^2)$ and $O(n^2)$, which yields a total complexity of $O(nm^2 + n^2)$.

Changing $T_{i,i,\mathbf{A}}$ is accomplished by performing a line search with a fixed number of iterations. During every iteration, a new area $\mathbf{A}_{\text{new}}$ is computed ($O(m)$), and it is tested whether $\mathbf{A}_{\text{new}}$ is a subset of $T_{i,i,\mathbf{B}}$ ($O(m^2)$). In order to be an acceptable solution, $\mathbf{A}_{\text{new}}$ has to avoid certain areas, according to the rules (2.2) through (2.4). This is ensured by testing whether $\mathbf{A}_{\text{new}}$ intersects any of the forbidden areas, every check being of order $O(m^2)$. There are $O(n)$ checks total (one for each agent j), which yields a total complexity of $O(nm^2)$.

The total complexity of one frame of the entire algorithm is $O(nm^2 + n^2)$. It follows that the number of line segments m used to describe the areas should be kept low. One possible way to accomplish this is to limit the maximum number of line segments used to approximate the paths of the agents and tolerate the resulting approximation errors. The computational complexity is then only a function of the number of agents. Note that the only operation that is $O(n^2)$ is the computation of the cost function, everything else is linear in n. In general, other cost functions can be chosen which scale linearly in n, depending on the application and the performance desired.

Large number of agents n

If the total number of agents n is very large the computation times can become prohibitively long. One way to avoid this is to reduce the number of agent interactions by

limiting the wireless communication. Assume that two agents can only communicate if they are closer than the maximum wireless range R_{wl}. Define n_2 as the number of agents which are in wireless range of agent i. The basic algorithm requires i to communicate with all n_2 agents, therefore the computational complexity of i is $O(n_2 m^2 + n_2^2)$. If the agent distribution is uniform such that every agent can communicate with n_2 other agents the computational complexity in general is $O(n_2 m^2 + n_2^2)$. Hence the complexity is independent of the total number of agents n, and n_2 becomes a characteristic parameter of the system.

The path-planning algorithm requires that an agent has to be able to communicate with every agent that it could possibly collide with. In order to guarantee safety in the presence of limited wireless range an agent i can only request areas which lie inside a circle with radius $R_{\text{wl}}/2$, centered about $\mathbf{p}_i$. This avoids the intersection of requested areas of two agents which cannot communicate. If the task location is farther than $R_{\text{wl}}/2$ from $\mathbf{p}_i$ then i has to set $\mathbf{f}_i$ to a temporary location on the path from $\mathbf{p}_i$ to $\mathbf{f}_i$, such that $\|\mathbf{f}_i - \mathbf{p}_i\| < R_{\text{wl}}/2$. Upon reaching $\mathbf{f}_i$ i will have to negotiate the remaining path to the task location.

Another requirement of the path-planning algorithm is that every agent i has to know which other agents j are in range and could possibly cause a conflict, since during the handshake i has to contact every j. In case of limited wireless range this requirement can be realized by utilizing a grid-based approach, for example: Divide the area on which the agents move into grids and provide every grid with a server that can communicate with the agents. Every server stores which agents are present in that particular grid. Agents are responsible for contacting the server and announcing their presence as well as removing their identification number from the server once they leave the grid. Before every handshake an agent has to contact the nearest server in order to determine which

other agents it has to communicate with.

2.4 Performance Improvement Using Waypoints

The response of the basic algorithm to prevent imminent collisions is to stop the agents until the obstacle has been removed. However, agents could have to wait a significant amount of time to let another agent pass. In extreme cases two agents could be waiting for each other indefinitely, for example if their initial positions, temporary and final destinations are located on a single line, such that the agents would have to move through each other in order to get to their respective destinations. For these reasons an extension to the basic algorithm is proposed, which improves performance and prevents deadlocks.

The idea is to treat the final destination $\mathbf{f}_i$ as a waypoint. The actual task location is then denoted by $\mathbf{g}_i$. The final destination can be adjusted freely in order to maneuver around obstacles while moving towards $\mathbf{g}_i$. These adjustments can be made while the agents are in motion, which allows them to react quickly to the current situation. Safety will still be guaranteed due to the nature of the basic algorithm which handles the reservation of areas. There is a trade-off between performance and increased wireless communication, since in general a change of $\mathbf{f}_i$ requires a new handshake. Figure 2.6 depicts an example where the direct path between i and $\mathbf{g}_i$ is blocked. Agent i sets a sequence of waypoints, $\mathbf{f}_{i,1}$ and $\mathbf{f}_{i,2}$, which guide it around the obstacle.

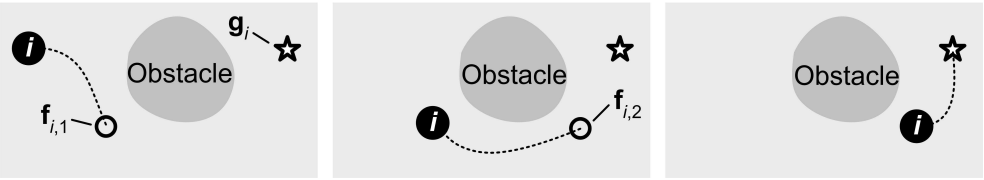

Figure 2.6: Example of using waypoints

The waypoint locations are chosen according to heuristic guidelines. The following paragraphs describe one possible approach where an attempt is made to minimize the average time required for the agents to reach their task locations. In order to use waypoints line pc:4 of the pseudo-code has to be replaced with the following instructions:

1. IF ($\mathbf{g}_i$ has been reached)
2. Set a new $\mathbf{g}_i$, $\mathbf{f}_i = \mathbf{g}_i$
3. ELSE
4. IF ($\mathbf{f}_i$ has been reached)
5. $\mathbf{f}_i = \mathbf{g}_i$
6. ELSE
7. Evaluate situation, adjust $\mathbf{f}_i$ as desired

Note that during step 4 tolerances can be much looser than during step 1. In general, this will result in higher performance, since the agent will proceed to the next waypoint without having to come to a full stop at $\mathbf{f}_i$. On line 7 the waypoint $\mathbf{f}_i$ is adjusted depending on the current situation and the chosen heuristic. In the presented implementation three frequently recurring situations are detected:

- The path of i leads through a requested area $T_{j,j,\mathbf{B}}$ and j has priority
- The path of i leads through a reserved area $T_{j,j,\mathbf{A}}$ and i has priority
- There exists a clear path from $\mathbf{p}_i$ to $\mathbf{g}_i$

In the first case i would have to wait until either $T_{j,j,\mathbf{B}}$ has been changed significantly or i gets priority. Most likely it will be faster, however, to set a waypoint $\mathbf{f}_i$ such that i takes a detour around $T_{j,j,\mathbf{B}}$. The waypoint $\mathbf{f}_i$ is found by determining the closest point

to the line through $\mathbf{p}_i$ and $\mathbf{g}_i$, such that the paths from $\mathbf{p}_i$ to $\mathbf{f}_i$ and $\mathbf{f}_i$ to $\mathbf{g}_i$ do not intersect the requested area of j.

In the second case j is supposed to yield to i by changing $T_{j,j,\mathbf{A}}$ such that it does not intersect $T_{j,i,\mathbf{B}}$ anymore. For some reason this is not happening, for example because agent dynamics prevent j from stopping early. Agent i can either wait or attempt to move around $T_{j,j,\mathbf{A}}$. In the current implementation i will try several random waypoints until it finds one that yields a path that does not intersect $T_{j,j,\mathbf{A}}$. It is also possible to do a more structured search like in case 1, but it turns out that in practice case 2 occurs when the agent is already very close to the blocking reserved area. Randomizing the waypoints gets the agent moving, which is then able to move around the obstacle.

In the third case, the waypoint is set directly to $\mathbf{g}_i$, which results in the fastest possible path to the task location.

2.5 Results

The presented algorithm is implemented in C++ and tested in simulation and on a real system of autonomous robots. While safety is guaranteed, performance is dependent on e.g. the cost function, wireless latency, the kind of waypoints that are being used, and the agent density. Five agents were simulated on a 4.9 m by 3.4 m area. The maximum acceleration of every agent was set to 5 m/s^2, the maximum velocity to 2.3 m/s. These specifications stem from the actual Cornell RoboCup platform, see below. For the given case one frame of the main loop executes in approximately 0.55 ms on a 1.7 GHz Pentium 4. This makes the algorithm suitable for application in a real-time environment with fast dynamics.

The performance metric is defined as the average velocity of an agent towards its

task location. Different scenarios were tested, varying the latency of the wireless transmission and the waypoints used. The task locations were selected randomly, without loss of generality. The results are presented in Table 2.2, where n denotes the number of agents, L denotes wireless latency (time from sending a packet to receiving it), #Tasks stands for the total number of task locations that were visited, v_{av} is the average velocity towards the targets, and P_{av} is the average number of packets sent per frame by each agent. The last row of Table 2.2 contains the simulation results using a single agent. This is the theoretical upper limit of the performance since in that case no communication is required and the agent can move directly from task location to task location. As expected, performance degrades as the wireless latency increases. This is

Table 2.2: Results

n	Waypoints?	L [ms]	#Tasks	v_{av} [m/s]	P_{av}
5	no	16	1047	0.178	0.15
5	no	80	1003	0.159	0.12
5	no	160	1021	0.147	0.11
5	yes	16	1026	0.789	0.39
5	yes	80	1002	0.564	0.30
5	yes	160	1000	0.375	0.25
1	-	-	1080	1.301	-

partly caused by the additional delay itself. However, the secondary effects are more severe: The entire information flow slows down, as indicated by the decreased number of sent packets P_{av}. The agents are slower to react to changing situations, less capable of avoiding choke points, and more prone to stop. A motionless group of agents forms an even larger obstacle and is more likely to affect other agents, causing a chain reaction.

A significant performance improvement can be achieved by using waypoints. The agents move around restricted areas altogether instead of waiting until the obstacle disappears. The trade-off is increased wireless communication, since agents have to reserve and request areas more frequently.

The algorithm was also successfully implemented on the 2005 Cornell RoboCup platform. The system consists of five omnidirectional robots equipped with PC104 single-board computers and 802.11a wireless communication systems. Localization on the field (5.5 m x 4.0 m) is accomplished by a vision system comprised of two overhead cameras and a vision computer. Prediction, local control, and motion primitives from the RoboCup system are used to provide basic motion functionality, see [37]. The path-planning algorithm runs at a rate of 60 Hz, which is also the update rate of the vision information. Maximum acceleration and velocity of the agents are the same as in simulation (5 m/s^2, 2.3 m/s). The simulation results are backed up by the experiments: the algorithm provides collision-free trajectories to the desired destinations in real-time.

Video clips of the algorithm applied to the RoboCup system can be found at the author's web site [64], as well as the source code of the algorithm in C++.

2.6 Conclusion and Future Prospects

A cooperative decentralized path-planning algorithm for a group of autonomous agents has been presented, which provides guaranteed collision-free trajectories in real-time in the presence of arbitrary wireless delay. Safety is maintained by reserving exclusive areas for each agent. A handshaking procedure guarantees that the agents have recent information when interacting. Conflicts between agents are resolved using a cost-based negotiation process. The underlying mathematical operations lend themselves naturally to the application of computational geometry. The algorithm was implemented

and tested successfully in simulation and on a system of real robots.

The building blocks of the proposed algorithm are flexible with respect to different underlying systems. Agent dynamics are captured by a modular motion primitive, while the cost function and rules to generate waypoints can easily be adapted to the system at hand. One possible implementation has been presented featuring five omnidirectional agents moving to task locations on a 2D surface.

In general, the more accurately the requested/reserved areas represent the dynamics of the agents, the higher the performance of the algorithm. A performance improvement could be achieved by allowing non-convex $\mathbf{B}$, which would decrease intersections between requested areas at the price of increased computation.

In the presented approach it is assumed that interaction between the agents is limited to wireless communication, since the agents cannot localize each other. This assumption should be seen as the minimum requirement for the algorithm to function. In practice agents could have additional sensors with which to locate each other, for example to detect agents which are not operating according to the rules of the algorithm ("rogue agents").

The current version of the algorithm assumes that the agents can come to a full stop. However, the algorithm can be extended to vehicles which have to maintain a minimum velocity, such as airplanes. In that case agents would have to reserve areas which include enough room for a loiter pattern, as described for example in [26].

Chapter 3
Performing and Extending Aggressive Maneuvers using Iterative Learning Control

Abstract

This paper presents an algorithm to iteratively perform an aggressive maneuver, i.e. drive a system quickly from one state to another. A simple model which captures the essential features of the system is used to compute the reference trajectory as the solution of an optimal control problem. Based on a lifted domain description of that same model an iterative learning controller is synthesized by solving a linear least-squares problem. The controller adjusts a feedforward signal using the results of experiments with the system. The non-causality of the approach makes it possible to anticipate recurring disturbances. Computational requirements are modest, allowing controller update in real-time. The experience gained from successful maneuvers can be used to adjust the model, which significantly reduces transients when performing similar motions. The algorithm is successfully applied to a real quadrotor unmanned aerial vehicle. The results are presented and discussed.

3.1 Introduction

With the increasing application of autonomous systems there arises a need to take advantage of their full capabilities. Pushing the envelope of these vehicles invariably involves dealing with transients and nonlinear dynamics. One approach to improve the performance is to identify the system well and apply advanced control methods. However, the required level of accuracy could require extensive system identification efforts. Further, a model-based approach only works satisfactorily if all vehicles of a series have very similar dynamics, which could necessitate tight tolerances and expensive hardware.

A different paradigm is to put the complexity in the software and take advantage of the low cost of sensors. A relatively simple model in conjunction with an adaptive algorithm and a well-chosen set of sensors allows each vehicle to experimentally determine how to perform a difficult maneuver and to compensate for individual differences in the system dynamics. This data-based approach to control has been proposed by many authors.

Moore [59] presents an algorithm to control a robotic manipulator based on sab-trees (state, action, behavior). These data structures store the experimental results of the system, to be retrieved in real-time for control. Depending on the current state of the system and the desired behavior the matching action is chosen. Schaal and Atkeson [42] pursue a similar approach. Local regression is performed on the stored data, the results of which are used to compute a local model and controller. Hjalmarsson [43] and Jansson [44] describe iterative feedback tuning (IFT) which performs a stochastic gradient search on a performance metric of the system. The gradient of the performance is computed directly from input/output data by appropriate selection of the experimental input. Similar in spirit is the one-shot method virtual reference feedback tuning (VRFT) [45] [46] which computes near-optimal controller parameters from input/output data.

Abbeel et al. [47] [48] demonstrate aerobatic maneuvers using an autonomous helicopter by application of reinforcement learning. A human expert flying the helicopter provides an initial guess of the desired trajectory, from which a model and a feedback controller are extracted. The differential dynamic programming (DDP) algorithm runs in real-time.

Another approach is called iterative learning control (ILC). The idea behind ILC is that the performance of a system executing the same kind of motion repeatedly can be improved by learning from previous executions. Given a desired output signal ILC algorithms experimentally determine an open-loop input signal which approximately

inverts the system dynamics and yields the desired output. Bristow et al. [40] provide a survey of different design techniques for ILC. The update of the input signal can be based on PD-type functions, which requires very little knowledge of the underlying plant dynamics. Plant inversion leads to fast convergence, but relies on a very accurate model. On the other hand, $\mathcal{H}_\infty$ based methods provide more robustness at the cost of performance. Another systematic option which requires a model is the minimization of a quadratic cost criterion.

Chen and Moore [41] present an approach based on local symmetrical double-integration of the feedback signal and apply it to a simulated omnidirectional ground vehicle. Two tuning parameters adjust the low-pass characteristics and the convergence rate of the ILC. Ghosh and Paden [58] show an approach based on approximate inversion of the system dynamics. Chin et al. [51] merge a model predictive controller [49] with an ILC [50]. The real-time feedback component of this approach is intended to reject non-repetitive noise while the ILC adjusts to the repetitive disturbance. Cho et al. [52] put this approach in a state-space framework.

Non-causal control laws allow ILC to preemptively compensate for disturbances or model uncertainties which are constant from trial to trial. Formulating the problem and controller in the lifted domain is a natural way of exploiting the repetitive nature of the experiments, made viable by advancements in computer processors and memory. Rice and Verhaegen [53] present a structured unified approach to ILC synthesis based on the lifted state-space description of the plant/controller system. The sequentially semi-separable structure of the problem is then exploited to synthesize the controller efficiently.

In practice ILC have been applied to repetitive tasks performed by stationary systems, such as wafer stages [54], chemical reactors [55], or industrial robots [56]. Appli-

cations to autonomous vehicles are more rare.

This paper presents a lifted domain ILC algorithm which enables a system to perform an aggressive motion, i.e. drive the system from one state to another. Aggressive in this context characterizes a maneuver that takes place in the nonlinear regime of the system and/or close to the state or input constraints. This maneuver would be hard to tune by hand or would require very accurate knowledge of the underlying system. Instead, the featured algorithm only requires a comparatively simple model (which captures essential system dynamics) and initial guess for the input. In case of an unstable system, it is assumed that a stabilizing controller is available. The feedforward signal required to perform the maneuver is iteratively determined using experiments.

While the algorithm is applicable to a wide range of systems, it is particularly intended for autonomous vehicles. Hence, a requirement is added that the controller update can be executed online with modest computational resources, putting emphasis on computational efficiency. In general the algorithm re-uses as much data as possible for the purposes of safety and efficiency. For example, the model used to stabilize the system initially is employed to determine the ILC update law. If a particular maneuver is performed satisfactorily the gained knowledge can be utilized to perform a maneuver which is similar to the one just learned. In this way a motion could be slowly extended, eventually executing a maneuver which could not have been performed given just the initial model.

The first step of the algorithm is the computation of the reference trajectory and input. This is accomplished by solving an optimal control problem based on the given model and constraints. The nonlinear model is then linearized about this reference and discretized, resulting in a discrete time linear time-varying (LTV) system. The lifted description of this LTV system defines the input-output relationship of the system for one

complete experimental run in form of a single matrix. After performing an experiment, the results are stored and compared with the ideal trajectory, yielding an error vector. Solving a linear least-squares (LLS) problem based on the lifted LTV system and the error vector yields the change in the input signal for the next trial. The algorithm terminates if the norm of the error vector is sufficiently small.

To the best of our knowledge the application to a real UAV and extension of the maneuver based on previous experiments makes this work an original contribution to the field.

The algorithm is successfully applied to an unmanned aerial vehicle (UAV). After stabilization, the UAV is a marginally stable system which requires accurate feedforward inputs to track a reference satisfactorily. This makes the UAV a challenging testbed to justify the chosen approach.

The rest of the paper is organized as follows: Section 3.2 presents the dynamics of the vehicle used for algorithm derivation and implementation. Section 3.3 describes the algorithm to perform a single maneuver, consisting of reference generation and update laws for the control input. The learned maneuver is extended in Section 3.4, taking advantage of the previously collected information. Section 3.5 shows the successful application of the algorithm to a real rotorcraft, while Section 3.6 provides a conclusion and future prospects.

3.2 Vehicle Dynamics

The presented algorithm is being applied to the quadrotor UAV shown in Figure 3.1. The dynamics of the vehicle are unstable so to perform experiments with the airborne vehicle, an initial controller is required which stabilizes the UAV in hover. A PD con-

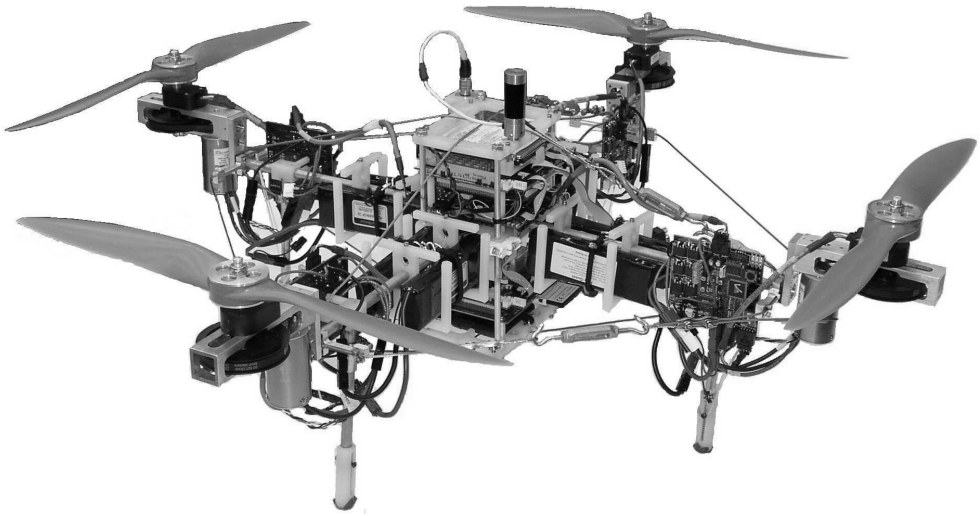

Figure 3.1: Quadrotor Unmanned Aerial Vehicle

troller has been synthesized based on the linearized six degrees of freedom (DOF) rigid body dynamics. In the following this controller will be referred to as hover controller. Note that the hover controller is only active before and after an ILC run (to bring the vehicle into position), not during a trial. For a more extensive description of the vehicle and the local controller see Appendix D.

The maneuver of interest in this paper consists of a sideways motion in the xz plane, see Figure 3.2 (left). This reduces the dynamics to two-dimensional space and three DOF without loss of generality. Figure 3.2 (right) shows the free-body diagram of the equivalent 2D vehicle. The global frame of reference (FOR) is denoted by x, z, and θ. Thrust vectors f_0 and f_1 extend from the centers of the propellers and are offset from the vertical vehicle axis by θ_δ, which represents imperfectly aligned actuators, for example. The distances between thrust and center of mass are denoted by l_0 and l_1. Note that gravity g acts in positive z direction. The vehicle mass is denoted by m, the moment of inertia by j.

As a propeller spins, it generates lift (thrust) and drag. The hardware determines the

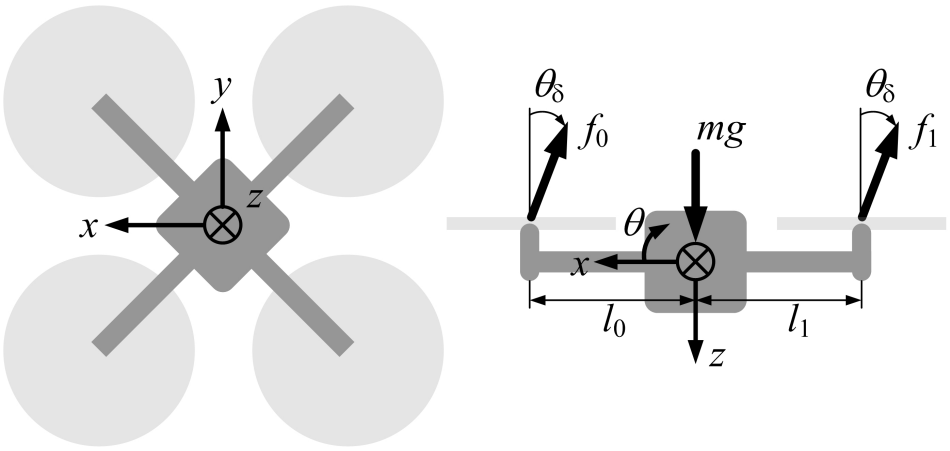

Figure 3.2: UAV Top Down View (left), Free-body Diagram (right)

maximum thrust $f_{i,max}$ and maximum rate of change of thrust $\dot{f}_{i,max}$, since every change of thrust requires a change of the propeller rpm. Table 3.1 contains the parameters of the UAV.

Table 3.1: System parameters of the UAV

g	m	j	l_0	l_1	$f_{i,max}$	$\dot{f}_{i,max}$
9.81 m/s^2	5.6 kg	0.1593 m^{2}kg	0.2623 m	0.2623 m	33.3 N	12 N/s

Taking the force and moment balances yields the open-loop (no hover controller) equations of motion of the rigid body in the global FOR.

$$\begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{1}{m}\sin(\theta+\theta_\delta)(f_0+f_1) \\ -\frac{1}{m}\cos(\theta+\theta_\delta)(f_0+f_1) \\ \frac{1}{j}(f_0 l_0 - f_1 l_1) \end{bmatrix} + \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} \tag{3.1}$$

As mentioned before, any kind of more sophisticated dynamics such as aerodynamics or flexible structures are neglected for the sake of simplicity. To stabilize the system

and reduce the impact of nonlinearities, a control loop is implemented for the rotation such that the response of θ is that of a second order system

$$\ddot{\theta} = -2\zeta\omega_n\dot{\theta} - \omega_n^2(\theta - \theta_c) \tag{3.2}$$

$$= k_{\dot{\theta}}\dot{\theta} + k_\theta(\theta - \theta_c) \tag{3.3}$$

with damping ratio ζ, natural frequency ω_n, and commanded angle θ_c. The parameters ζ and ω_n are selected to match the system response of the closed loop system with the hover controller in place. This guarantees stability and adequate performance of the θ stabilization in hover. Note that this loop only stabilizes the rotation during the ILC trials, while the hover controller stabilizes all six degrees of freedom whenever the vehicle is not executing the ILC. Combining the thrust inputs f_0 and f_1 to be

$$f_a = f_0 + f_1 \tag{3.4}$$

the stabilized equations of motion of the vehicle then become

$$\begin{bmatrix} \ddot{x} \\ \ddot{z} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -\frac{1}{m}\sin(\theta + \theta_\delta) f_a \\ -\frac{1}{m}\cos(\theta + \theta_\delta) f_a \\ k_{\dot{\theta}}\dot{\theta} + k_\theta(\theta - \theta_c) \end{bmatrix} + \begin{bmatrix} 0 \\ g \\ 0 \end{bmatrix} = \mathfrak{f}(\rho, q, u) \tag{3.5}$$

where f_a and θ_c are the new system inputs. The state vector $q(t)$ is defined to be

$$q(t) = \begin{bmatrix} x(t) & z(t) & \dot{x}(t) & \dot{z}(t) & \theta(t) & \dot{\theta}(t) \end{bmatrix}^T \tag{3.6}$$

the input $u(t)$ is defined as

$$u(t) = \begin{bmatrix} f_a(t) & \theta_c(t) \end{bmatrix}^T \tag{3.7}$$

while the parameter vector ρ is defined as

$$\rho = \begin{bmatrix} m & l_0 & l_1 & \theta_\delta \end{bmatrix}^T \tag{3.8}$$

For specifics about the selection of the parameter vector, see Section 3.4.

3.3 Performing a Maneuver

The objective of the presented algorithm is to perform a maneuver from an initial state q_0 to a target state q_f. For the purpose of this paper, q_0 and q_f are both equilibria, i.e. the vehicle is hovering with $\dot{q} = 0$ in both cases. Note that this is not a requirement of the algorithm. However, this selection is advantageous due to better controlled initial conditions. Further, safety is increased since the vehicle is not moving in q_0 and q_f and therefore not endangering itself or the environment. Performing a maneuver consists of two parts: generation of the reference trajectory $q_d(t)$ and learning how to track it. The approach of tracking $q_d(t)$ covers two cases, deterministic and stochastic noise.

3.3.1 Generation of a reference trajectory

For some applications $q_d(t)$ and $u_d(t)$ may be known ahead of time, for example when learning from an expert. For the system at hand, the reference trajectory $q_d(t)$ is the solution of an optimal control problem (OCP): compute the minimum time solution

$$(q_d(t), u_{d,temp}(t)), \quad u_{d,temp}(t) = \left[f_{0,d}(t)\ f_{1,d}(t)\right]^T \tag{3.9}$$

which drives the system (3.1) from the initial state $q(0) = q_0$ to the final state $q(t_f) = q_f$, subject to constraints on the control effort

$$|f_i(t)| \leq f_{i,max} \tag{3.10}$$

$$|\dot{f}_i(t)| \leq \dot{f}_{i,max} \tag{3.11}$$

For the purpose of this paper, the OCP is solved using RIOTS [61], an optimal control toolbox written in Matlab and C. The above formulation of the OCP has been chosen for simplicity in the expressions of the constraints, which benefits the numerical solution process of the OCP. However, in subsequent parts of the algorithm the inputs

according to system (3.5) are being used. Therefore the optimal inputs $u_{d,temp}(t)$ must be transformed to $u_d(t) = \left[f_{a,d}(t)\ \theta_{c,d}(t)\right]^T$, such that $u_d(t)$ applied to (3.5) yields $q_d(t)$. Substituting (3.3) into the third line of (3.1) yields

$$\frac{1}{j}(f_0 l_0 - f_1 l_1) = k_{\dot{\theta}}\dot{\theta} + k_\theta(\theta - \theta_c) \tag{3.12}$$

which can be solved for θ_c. Together with (3.4) this yields the new inputs

$$f_{a,d}(t) = f_{0,d}(t) + f_{1,d}(t) \tag{3.13}$$

$$\theta_{c,d}(t) = \frac{k_{\dot{\theta}}\dot{\theta}_d(t) + k_\theta\theta_d(t) - \frac{1}{j}\left[f_{0,d}(t)l_0 - f_{1,d}(t)l_1\right]}{k_\theta} \tag{3.14}$$

Note that k_θ is not equal to zero if the control loop around θ has a proportional term, which is the case for the application at hand.

3.3.2 Tracking the reference trajectory

One assumption of the algorithm is that the motion of the vehicle stays close to the generated reference trajectory $q_d(t)$ and input $u_d(t)$. Linearizing (3.5) about this trajectory and input yields

$$\dot{\tilde{q}}(t) = \left.\frac{\partial \mathrm{f}}{\partial q}\right|_{q_d,u_d} \tilde{q}(t) + \left.\frac{\partial \mathrm{f}}{\partial u}\right|_{q_d,u_d} \tilde{u}(t) \tag{3.15}$$

$$= A(t)\tilde{q}(t) + B(t)\tilde{u}(t) \tag{3.16}$$

with $q = q_d + \tilde{q}$ and $u = u_d + \tilde{u}$. Converting to a discrete time system results in a linear time-varying system

$$\tilde{q}(k+1) = A_D(k)\tilde{q}(k) + B_D(k)\tilde{u}(k) \tag{3.17}$$

with k denoting a discrete time step and N being the trial length in discrete time steps. The dynamics (3.17) of a complete trial are written in the lifted domain to exploit the

repetitiveness of the experiments and synthesize a non-causal controller

$$\tilde{Q} = P\tilde{U} \tag{3.18}$$

$$\begin{bmatrix} \tilde{q}(1) \\ \tilde{q}(2) \\ \vdots \\ \tilde{q}(N) \end{bmatrix} = \begin{bmatrix} B_D(0) & 0 & \cdots & 0 \\ A_D(1)B_D(0) & B_D(1) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ \Phi(N-1,1)B_D(0) & \cdots & & B_D(N-1) \end{bmatrix} \begin{bmatrix} \tilde{u}(0) \\ \tilde{u}(1) \\ \vdots \\ \tilde{u}(N-1) \end{bmatrix} \tag{3.19}$$

$$\Phi(n_1, n_2) = \prod_{\nu=n_1}^{n_2} A_D(\nu) \tag{3.20}$$

where $\Phi(n_1, n_2)$ denotes the state transition matrix from n_2 to n_1, the capital letters $\tilde{Q}$ and $\tilde{U}$ indicate lifted versions of $\tilde{q}$ and $\tilde{u}$ while P denotes the matrix containing the lifted dynamics. A widely used ILC approach [40] of changing the input takes the form

$$U_{j+1} = L_1 \left[U_j + L_2 E_j \right] \tag{3.21}$$

where the index j denotes the trial, L_1 and L_2 denote two filter functions (matrices), and E_j is the lifted error signal

$$E_j = Q_d - Q_{j,m}, \quad Q_{j,m} = Q_j + noise \tag{3.22}$$

with $Q_{j,m}$ representing noisy measurements of the state during a trial. The presented ILC takes advantage of the given model which captures the essential dynamics of the underlying system. The exact formulation of the update law depends on the assumptions made about the noise. In case that the noise $d(k)$ does not change from trial to trial the system takes the form

$$q_{j,m}(k) = q_d(k) + \tilde{q}_j(k) + d(k) \tag{3.23}$$

$$Q_{j,m} = Q_d + \tilde{Q}_j + D \tag{3.24}$$

with D being the lifted constant disturbance vector, representing repeatable modeling errors or repeatable process noise for example. Using (3.18) and (3.22) it follows that

$$Q_{j,m} = Q_d + P\tilde{U}_j + D \tag{3.25}$$

$$E_j = Q_d - Q_{j,m} = -P\tilde{U}_j - D \tag{3.26}$$

Performing a single experiment or trial with $\tilde{U}_0 = 0$ yields an error signal of

$$E_0 = -D \tag{3.27}$$

The input which minimizes the square of the error signal is the solution of a linear least-squares problem:

$$\tilde{U}_1 = \arg\min_{\tilde{U}} \|E\|_2^2 = \arg\min_{\tilde{U}} \|P\tilde{U} + D\|_2^2 \tag{3.28}$$

$$= -P^{\dagger}D \tag{3.29}$$

$$= \tilde{U}_0 + P^{\dagger}E_0 \tag{3.30}$$

where $P^{\dagger}$ indicates the pseudo-inverse

$$P^{\dagger} = \lim_{\epsilon \to 0} (P^T P + \epsilon I)^{-1} P^T, \quad \epsilon > 0 \tag{3.31}$$

which can be computed by well-established methods such as singular value decomposition (SVD) [38]. The input $\tilde{U}_1$ applied to the same system (3.25), will result in an error of

$$E_1 = -(I - PP^{\dagger})D \tag{3.32}$$

The update law $P^{\dagger}$ can be non-causal, which means that the system will compensate for a trial-independent error occurring in the future. This results in a dense matrix $P^{\dagger}$. As an example, a part of the gain matrix $P^{\dagger}$ for system (3.5) executing maneuver 1 (see Section 3.5) has been visualized in Figure 3.3. The x-axis denotes the time index k_{err} of the rotation error $\theta_d(k_{err}) - \theta_m(k_{err})$, the y-axis denotes the time index k_{inp} of the desired change in collective thrust $f_a(k_{inp})$, and the z-axis denotes the gain from the rotation error to the thrust. This shows that errors occurring at time steps k_2 can affect inputs at previous time steps $k_1 < k_2$.

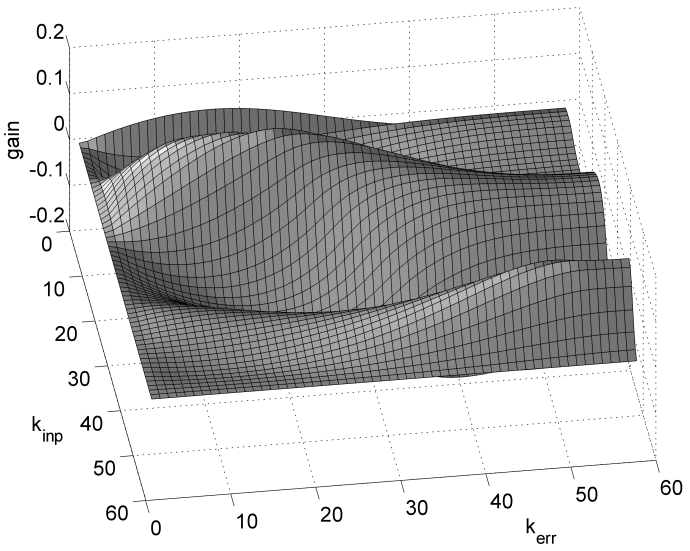

Figure 3.3: Partial Visualization of $P^{\dagger}$

In case when there is both a constant disturbance and additional white noise $v_j(k)$ that changes from trial to trial, the system takes the form

$$q_{j,m}(k) = q_d(k) + \tilde{q}_j(k) + d(k) + v_j(k) \tag{3.33}$$

$$Q_{j,m} = Q_d + \tilde{Q}_j + D + V_j \tag{3.34}$$

$$E_j = Q_d - Q_{j,m} = -P\tilde{U}_j - D - V_j \tag{3.35}$$

with V_j being the lifted noise. All components of $v_j(k)$ and V_j are assumed to be independent and identically distributed (iid) zero-mean Gaussian white noise. Using a similar approach as (3.30) for the update law results in

$$\tilde{U}_{j+1} = \tilde{U}_j + \alpha P^{\dagger} E_j, \quad \alpha \in (0, 1) \tag{3.36}$$

$$= \tilde{U}_j + L_2 E_j \tag{3.37}$$

with α being a tuning parameter. To show the impact of α in the case of many iterations, (3.35) is substituted into (3.36), such that the input dynamics in the trial domain become

$$\tilde{U}_{j+1} = (I - \alpha P^{\dagger} P)\tilde{U}_j - \alpha P^{\dagger} D - \alpha P^{\dagger} V_j \tag{3.38}$$

$$= (1-\alpha)\tilde{U}_j - \alpha P^\dagger D - \alpha P^\dagger V_j \tag{3.39}$$

$$= (1-\alpha)[(1-\alpha)\tilde{U}_{j-1} - \alpha P^\dagger D - \alpha P^\dagger V_{j-1}] - \alpha P^\dagger D - \alpha P^\dagger V_j \tag{3.40}$$

$$= -\alpha[1 + (1-\alpha) + ... + (1-\alpha)^j]P^\dagger D$$

$$-\alpha P^\dagger [V_j + (1-\alpha)V_{j-1} + ... + (1-\alpha)^j V_0] \tag{3.41}$$

$$= -\alpha\left[\sum_{i=0}^{j}(1-\alpha)^i\right]P^\dagger D - \alpha P^\dagger \sum_{i=0}^{j}(1-\alpha)^{j-i}V_i \tag{3.42}$$

$$= -\left[1-(1-\alpha)^{j+1}\right]P^\dagger D - \alpha P^\dagger \sum_{i=0}^{j}(1-\alpha)^{j-i}V_i \tag{3.43}$$

while assuming $\tilde{U}_0 = 0$ (best guess at first iteration) and using the identity

$$(1-\alpha)\left[\sum_{i=0}^{j}(1-\alpha)^i\right] - \left[\sum_{i=0}^{j}(1-\alpha)^i\right] = [(1-\alpha) + ... + (1-\alpha)^{j+1}]$$

$$-[1 + ... + (1-\alpha)^j] \tag{3.44}$$

$$-\alpha\left[\sum_{i=0}^{j}(1-\alpha)^i\right] = -1 + (1-\alpha)^{j+1} \tag{3.45}$$

In the limit of the number of iterations tending towards infinity, the input and error become

$$\tilde{U}_\infty = \lim_{j\to\infty}\tilde{U}_j = -P^\dagger D - P^\dagger W \tag{3.46}$$

$$E_\infty = \lim_{j\to\infty}E_j = -[I - PP^\dagger]D - V_\infty + PP^\dagger W \tag{3.47}$$

$$W = \alpha\sum_{i=0}^{j}(1-\alpha)^{j-i}V_i \tag{3.48}$$

where W is a noise vector with properties

$$\mathrm{E}[W] = 0 \tag{3.49}$$

$$\mathrm{E}[WW^T] = \frac{\alpha^2}{1-(1-\alpha)^2}\mathrm{E}[VV^T] = \frac{\alpha}{2-\alpha}\mathrm{E}[VV^T] \tag{3.50}$$

The parameter α serves as a tuning parameter to regulate the influence of V_j. For α approaching 1, the variance of W is not reduced, it is the same as performing the

update only once. For α approaching zero, the solution tends towards the optimum, i.e. minimizes the variance of W. The smaller the variance of W the smaller the variance of $\tilde{U}_\infty$ and E_∞, since

$$\begin{aligned} \mathrm{E}[\tilde{U}_\infty] &= -P^\dagger D && (3.51) \\ \mathrm{E}[(\tilde{U}_\infty + P^\dagger D)(\tilde{U}_\infty + P^\dagger D)^T] &= P^\dagger W W^T (P^\dagger)^T && (3.52) \\ \mathrm{E}[E_\infty] &= -[I - PP^\dagger]D && (3.53) \\ \mathrm{E}[(E_\infty + [I - PP^\dagger]D)(E_\infty + [I - PP^\dagger]D)^T] &= \mathrm{E}[VV^T] + PP^\dagger W W^T (PP^\dagger)^T && (3.54) \end{aligned}$$

However, in this case, the number of iterations required to achieve this solution tends towards infinity, because $\tilde{U}_{j+1}$ tends towards zero, meaning there is no update of the input. In practice the trade-off has to be somewhere in between.

In addition to the update law L_2, it is possible to introduce a low-pass filter in L_1 which rejects high frequency noise that is injected by the measurements.

The final algorithm takes the following form:

1. Define model, constraints, initial and final states (q_0, q_f)
2. Solve optimal control problem to find minimum time path from q_0 to q_f. Result: ($q_d(t)$, $u_d(t)$)
3. Linearize model about ($q_d(t)$, $u_d(t)$), discretize, and build lifted system model P
4. Set $j = 0$, $\tilde{U}_j = 0$
5. Run experiment. Input: $U_j = U_d + \tilde{U}_j$, output: Q_j
6. Compute error $E_j = Q_d - Q_j$
7. Stop if E_j below threshold
8. Update input $U_{j+1} = L_1 \left[U_j + L_2 E_j \right]$

9. $j = j + 1$, go to 5

The algorithm terminates successfully if in step 7 the error E_j is smaller than a specified threshold. In that case the final input is denoted $\tilde{U}_M$ and the final experimental trajectory is denoted $\tilde{Q}_M$.

3.4 Extending the Maneuver

In the previous section, an algorithm was presented which iteratively converges to track a given trajectory. In this section, a method is proposed to facilitate the tracking of $q_{d,2}(t)$, assuming that it is already known how to track $q_{d,1}(t)$, with $q_{d,1}(t)$ and $q_{d,2}(t)$ being similar. The second subscript denotes the particular maneuver. The most straightforward approach to track $q_{d,2}(t)$ is the direct application of the algorithm from Section 3.3. However, this would neglect valuable information gained from previous experiments. A better method is to utilize the final input $\tilde{U}_{M,1}$ and trajectory $\tilde{Q}_{M,1}$ from tracking the previous trajectory $q_{d,1}(t)$ in order to provide better initial guesses for the tracking of $q_{d,2}(t)$. The approach described here involves the adjustment of the model parameters ρ (3.8). The goal is to adjust the nonlinear model (3.5) s.t. this model integrated with the real input matches the real output, i.e.

$$q_{M,1}(k) \quad = \quad q_{th,1}(k), \quad k \in [1, N] \tag{3.55}$$

with

$$q_{th,1}(k) \quad = \quad \int_0^{t(k)} \mathfrak{f}(\rho, q(\tau), u_{M,1}(\tau))d\tau, \quad q(0) = q_{M,1}(0) \tag{3.56}$$

Using lifted vectors, this can be posed as a nonlinear quadratic optimization problem

$$\rho^* \quad = \quad \arg\min_{\rho} \left\| Q_{M,1} - Q_{th,1} \right\|_2^2 \tag{3.57}$$

The optimal ρ^* is then substituted into the model (3.5) to provide the basis for the algorithm as described in Section 3.3.

The selection of adjustment parameters can affect the convergence of the optimization process of (3.57) involving a nonlinear system. The specifics of the system determine which set of parameters has significant impact on the behavior. The particular parameter vector (3.8) has been chosen since it allows the adjustment of the relationships between inputs and states. Further, it provides good results in practice, see Section 3.5. However, it should be noted that this selection is not unique and that other parameter vectors could provide similar results.

The easiest approach to parameter identification is to define a ρ which is valid over the entire duration of the trial. However, to reduce the residual of (3.57) N_ρ different parameter sets ρ_n are defined which are valid during consecutive intervals $[k_{\rho,n,0}, k_{\rho,n,f}]$ of equal size Δk_ρ. The subscript ρ indicates that the time index k refers to a parameter set, n denotes the number of the parameter set, and 0 and f denote the beginning and end respectively.

$$k_{\rho,n,f} + 1 = k_{\rho,n+1,0} \tag{3.58}$$

$$k_{\rho,n,f} - k_{\rho,n,0} = \Delta k_\rho \tag{3.59}$$

Figure 3.4 shows a plot of the residual of the optimization (3.57) over the number of parameter sets N_ρ for a typical experimental result $(\tilde{Q}_M, \tilde{U}_M)$. For the actual experiments N_ρ was set to four. Figure 3.5 depicts the error $q_M(k) - q_{th}$ for unadjusted ρ_0 (left) and adjusted ρ^* (right) for $N_\rho = 4$. This shows that the experimental results can be explained well by adjusting the parameter vector ρ. Further, the values of ρ_n^* are not unreasonably different from the unadjusted ρ_0, as can be seen in Table 3.2. There are some deviations in l_0 and l_1 which indicate that the vehicle requires more thrust to rotate, or that the

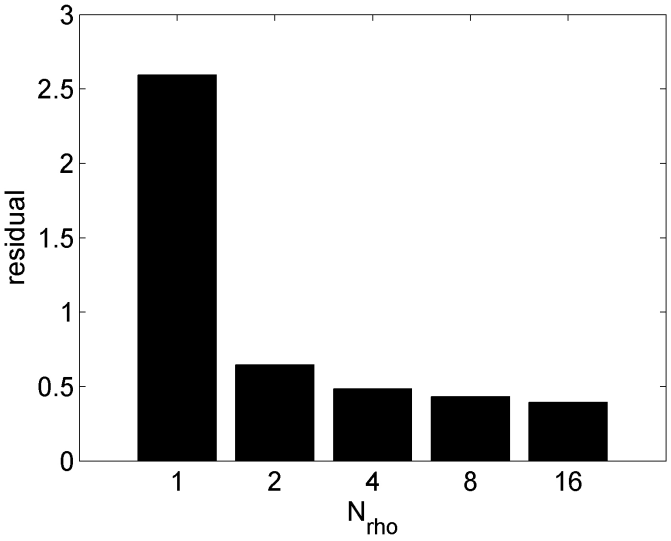

Figure 3.4: Residual of Optimization over Number of Parameter Sets

propellers do not produce as much thrust as expected. The deviation in θ_δ could stem from the fact that the vehicle is moving sideways, which changes the angle of attack of the propellers. In general, however, the parameter identification is masking a number of effects.

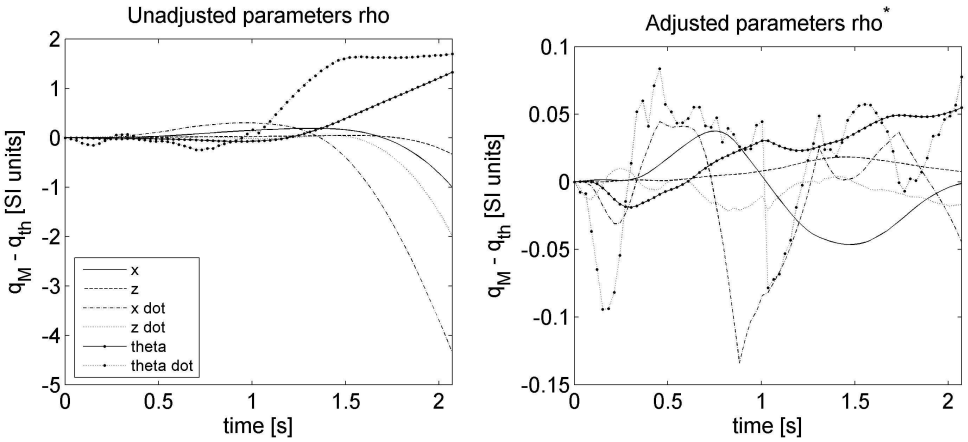

Figure 3.5: Theoretical Trajectory Error

Table 3.2: Comparison of ρ

Parameter	m [kg]	l_0 [m]	l_1 [m]	θ_δ [rad]
ρ_0	5.6	0.2623	0.2623	0
ρ_1^*	5.6318	0.2203	0.2233	-0.0054
ρ_2^*	5.4785	0.2059	0.1985	-0.0343
ρ_3^*	5.4744	0.2023	0.1973	0.1308
ρ_4^*	5.5770	0.1729	0.1649	0.0363

3.5 Implementation and Results

The algorithm was applied to the UAV shown in Figure 3.1. The iterative part of the ILC computations was implemented in C++ and executed while the vehicle was airborne. The computation of a single ILC iteration involved nonlinear transformations from 3D measurements to the 2D problem, low-pass filtering, and solving the LLS problem (3.28) by applying SVD. One iteration took about 5 - 10 seconds to complete on a 650 MHz Pentium processor. Therefore, it was possible to perform all iterations necessary for successful termination of the algorithm during a single flight without having to land and recharge the batteries. The rest of the algorithm, such as solving the OCP and adjusting the parameter vector, was implemented in Matlab since there was no need for online computation.

A side-to-side motion was selected as a characteristic trajectory to show the capabilities of the algorithm. Hovering at position q_0 the vehicle was required to move to position q_f as quickly as possible. This motion involved a significant amount of nonlinearities due to the quick acceleration and deceleration phases. Further, the banking of the vehicle in combination with high linear velocities was likely to introduce complex

fluid dynamic effects. These effects were not explicitly modeled, but expected to be part of the disturbance D, which the algorithm had to compensate for.

Four maneuvers were executed. The second subscript of q or u indicates the maneuver. The maneuvers are getting progressively more aggressive/faster. However, maneuvers 2 and 3 are identical, the difference is in the approach on how to compute the initial guess and the update law.

3.5.1 Maneuver 1

For the first maneuver initial and final states were defined as

$$q_{0,1} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.60}$$

$$q_{f,1} = \begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.61}$$

Solving the OCP yielded the desired reference trajectory $q_{d,1}(t)$ and input $u_{d,1}(t)$ shown in Figure 3.6.

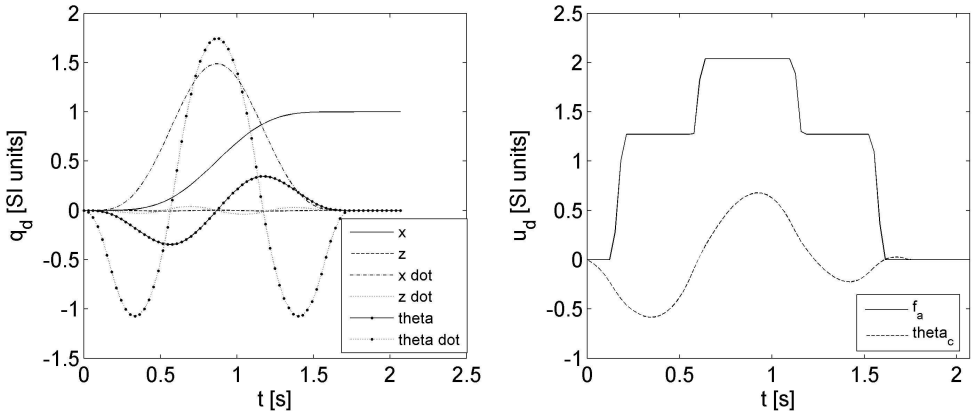

Figure 3.6: Maneuver 1, Solution of OCP: $q_{d,1}(t)$ (left), $u_{d,1}(t)$ (right)

The requirement for successful termination of the ILC was

$$\|q_{d,1}(N) - q_{M,1}(N)\|_2 \quad < \quad 0.2 \tag{3.62}$$

while the parameter α in the ILC update law was set to 0.3 to provide a trade-off between rejection of white noise and speed of convergence. Applying the algorithm to the real system led to convergence in 9 iterations, see Figure 3.7. Figure 3.8 depicts the error $q_{d,1}(t) - q_{j,1}(t)$ for different iterations j, visualizing the convergence of each state.

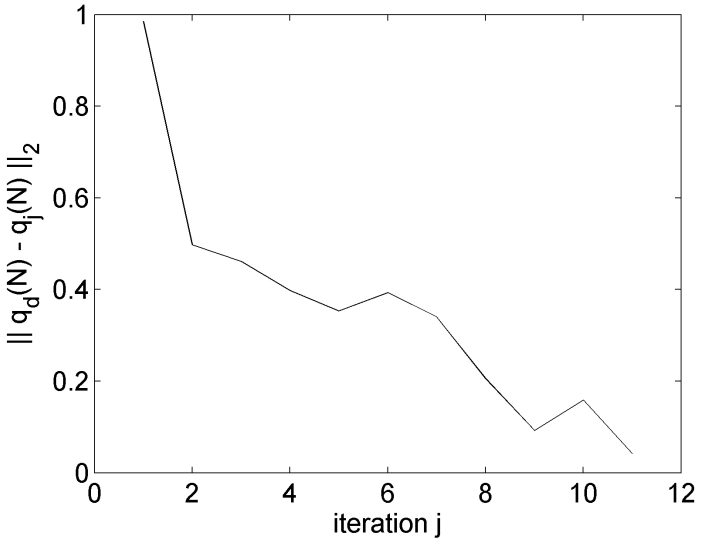

Figure 3.7: Maneuver 1, Error Norm over Iteration

3.5.2 Maneuver 2

The next maneuver was a side-to-side motion over 1.5 m, intended to excite more nonlinear behavior of the system.

$$q_{0,2} \quad = \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.63}$$

$$q_{f,2} \quad = \quad \begin{bmatrix} 1.5 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.64}$$

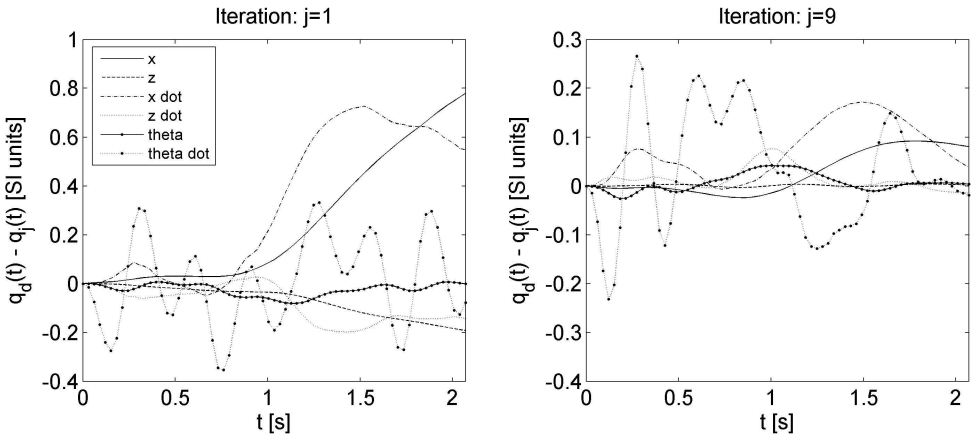

Figure 3.8: Maneuver 1, State Error

Learning this maneuver without utilizing any previous knowledge from maneuver 1 resulted in successful termination after 8 iterations, which is comparable to maneuver 1. However, the initial transients were more severe. The vehicle came close to becoming unstable or colliding with the boundaries of the airspace. Figure 3.9 shows the error $q_{d,2}(t) - q_{j,2}(t)$ for different iterations j.

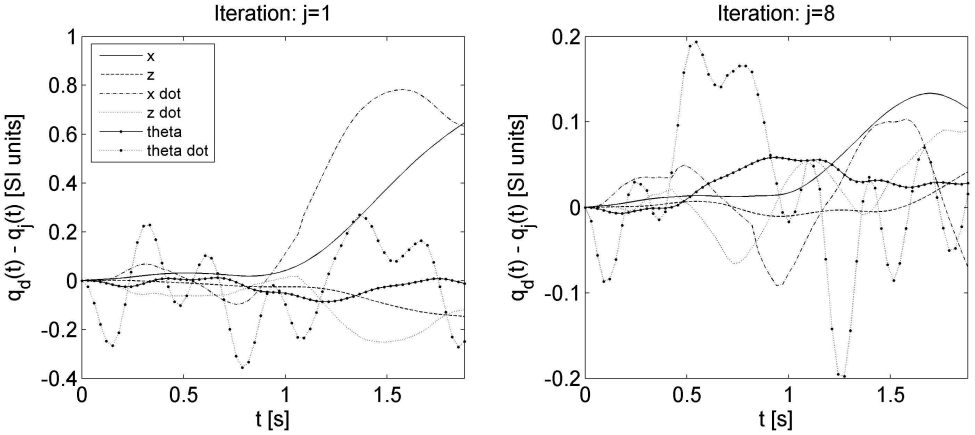

Figure 3.9: Maneuver 2, State Error

3.5.3 Maneuver 3

Maneuver 3 had the same initial and final conditions as maneuver 2. However, the initial guess of the input and the ILC update law were based on a model which used not the original parameters (3.8) but an identified set of parameters. Using the results from maneuver 1 and applying the approach described in Section 3.4 a new set of parameters was identified. Performing the 1.5 m side-to-side motion resulted in convergence after 7 iterations. While this small improvement in convergence time can be attributed to noise in the learning process, it should be noted that the initial transients were much smaller than for either maneuver 1 or 2, see Figure 3.10. This shows that it is beneficial to take advantage of previously gained information by adjusting the underlying model.

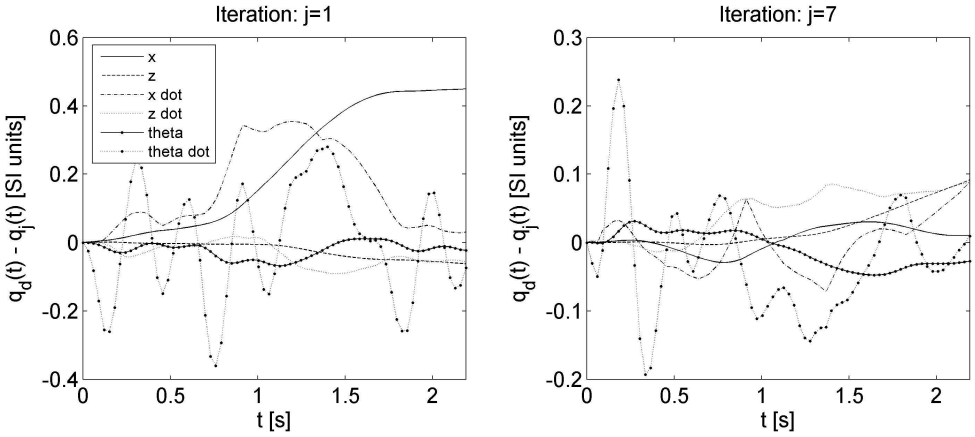

Figure 3.10: Maneuver 3, State Error

3.5.4 Maneuver 4

As a final test the results from maneuver 3 were used to adjust the model again with the goal of learning an even more aggressive maneuver 4, a 2.0 m side-to-side motion. Since the transients of maneuver 2 were close to destabilizing the UAV it was deemed

too risky to try learning the 2.0 m motion without using previous knowledge.

$$q_{0,4} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.65}$$

$$q_{f,4} = \begin{bmatrix} 2.0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.66}$$

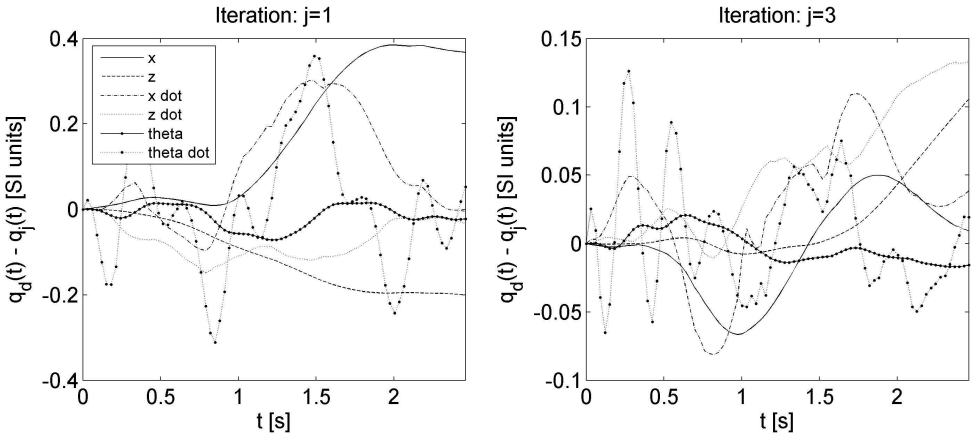

Figure 3.11: Maneuver 4, State Error

The results of the experiment in Figure 3.11 show that the initial transients are again much smaller than for either maneuver 1 or 2. The algorithm terminates successfully in only 3 iterations. The trends indicate that by reusing previous experimental data according to the approach suggested in Section 3.4 the number of required iterations and the initial transients can be reduced whenever trying to learn a similar maneuver.

3.6 Conclusion and Future Prospects

An algorithm has been presented which enables a system to iteratively learn and perform an aggressive motion, given a simple model which captures the essential dynamics of the system. Based on the model, an initial reference trajectory is computed as the solution of an optimal control problem. Expressing the problem in the lifted domain allows the synthesis of a non-causal controller, which can anticipate recurring

disturbances and compensate for them by adjusting a feedforward signal. The controller synthesis is formulated as a LLS problem, which can be readily solved and executed online with modest computational resources.

The algorithm has been successfully applied experimentally to a quadrotor UAV. Using the data from a well tracked trajectory, it is possible to adjust the model in order to learn a motion which is similar to the original reference. This approach reduces the required number of iterations and initial transients, enabling the execution of maneuvers which would be difficult to perform without previous knowledge. More complex motions could possibly be executed by training several basic motion primitives separately and then appending them.

As part of future research the LLS controller synthesis of the current algorithm can be refined by adding weights to some of the states, e.g. increasing the weights towards the final part of the motion. Further, rapid changes of the control input could be reduced by adding the derivative of the input to the LLS problem. A more systematic approach could involve expressing the controller update as a linear program, which would allow the explicit formulation of input and/or state constraints.

In case that the state of the system is not directly measurable the ILC update could involve a Kalman Filter to estimate the disturbance D. More complex noise models could then be implemented as well.

Appendix A
Assumption about Vehicle Weight Distribution

During the derivation of the vehicle equations of motion it is assumed that

$$n_1 + n_3 \quad = \quad n_2 + n_4 \tag{A.1}$$

This is motivated by the rigid pillar problem as presented in [16]. A vehicle with four wheels is statically undetermined if the wheels and suspension are considered to be rigid. In order to derive (A.1), the wheels are treated as linear springs with a very large spring constant k, such that $\ddot{z} = \ddot{\theta}_x = \ddot{\theta}_y = 0$ still holds to first order. The exact magnitude of the spring constant is not relevant (as long as it is positive and not infinite), since it will cancel out. The vehicle chassis is modelled as a rigid body, as shown in Figure A.1.

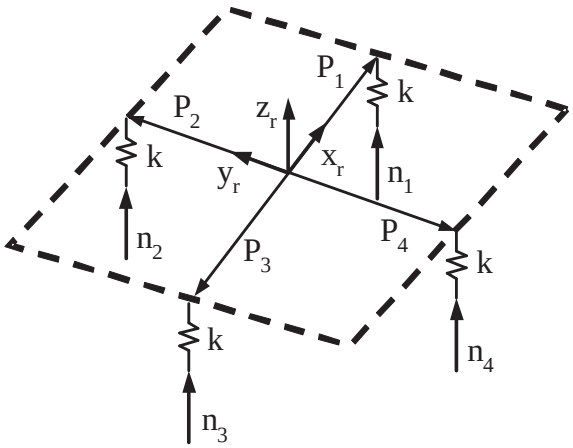

Figure A.1: Normal force model

For small rotations, the displacements d_i of the springs can be written as functions of the linear robot displacement z_r and the two rotations about the x_r and y_r axes, $\theta_{x,\mathrm{r}}$ and

$\theta_{y,\mathrm{r}}$ respectively.

$$d_1 = z_\mathrm{r} - l\theta_{y,\mathrm{r}} \tag{A.2}$$

$$d_2 = z_\mathrm{r} + l\theta_{x,\mathrm{r}} \tag{A.3}$$

$$d_3 = z_\mathrm{r} + l\theta_{y,\mathrm{r}} \tag{A.4}$$

$$d_4 = z_\mathrm{r} - l\theta_{x,\mathrm{r}} \tag{A.5}$$

$$kd_i = n_i \tag{A.6}$$

Solving (A.2) through (A.6) for the normal forces and adding n_1 and n_3 (n_2 and n_4 respectively) yields

$$n_1 + n_3 = 2kz_\mathrm{r} \tag{A.7}$$

$$n_2 + n_4 = 2kz_\mathrm{r} \tag{A.8}$$

which completes the derivation of (A.1).

Appendix B
List of Variables

Table B.1: List of variables

Variable	Unit	Description
$\mathbf{p}_i = (x_i, y_i)$	[m, m]	xy-position of agent i
$\mathbf{v}_i = (\dot{x}_i, \dot{y}_i)$	[m/s, m/s]	xy-velocity of agent i
$\mathbf{d}_i = (d_{x,i}, d_{y,i})$	[m, m]	xy-position of temporary destination of agent i
$\mathbf{f}_i = (f_{x,i}, f_{y,i})$	[m, m]	xy-position of final destination of agent i
$T_{i,j,\mathbf{A}}$	[ ]	information state of agent i, area **A** of agent j
$T_{i,j,\mathbf{B}}$	[ ]	area **B** of agent j
$T_{i,j,D}$	[ ]	information flag w.r.t. agent j
$T_{i,j,R}$	[ ]	priority w.r.t. agent j
$T_{i,j,R,\text{other}}$	[ ]	the last priority command to agent j
$T_{i,j,t1}$	[frames]	time of first packet of handshake
$T_{i,j,t2}$	[frames]	time when j sent the last packet that reached i
$T_{i,j,c}$	[ ]	cost of i w.r.t. j
$T_{i,j,c,\text{other}}$	[ ]	cost of j w.r.t. i
$T_{i,j,F}$	[ ]	1 if i and j are communicating, 0 otherwise
$T_{i,j,t5}$	[frames]	time when PA required other agent to give up R
$T_{i,j,\text{send}}$	[ ]	indicates whether i has to send a packet

Variable	Unit	Description
$P_{q,id}$	[ ]	packet number q, index of the sending agent
$P_{q,id,\text{other}}$	[ ]	index of the receiving agent
$P_{q,\mathbf{A}}$	[ ]	area **A** of the sending agent
$P_{q,\mathbf{B}}$	[ ]	area **B** of the sending agent
$P_{q,t,\text{me}}$	[frames]	time stamp of sending agent
$P_{q,t,\text{other}}$	[frames]	last time stamp from receiving agent
$P_{q,requD}$	[ ]	1 if a sender requires a response, 0 otherwise
$P_{q,c}$	[ ]	cost of sending agent w.r.t. receiving agent
$P_{q,R}$	[ ]	1 if sending agent has priority, 0 otherwise
$P_{q,R,\text{other}}$	[ ]	Priority command

Appendix C
Proof of Safety

In the following it is shown that the process of requesting and reserving areas guarantees safety, i.e., prevents collisions. Safety will be proved for a pair of agents i and j. The extension to n agents follows in a straightforward way by applying the proof to every pair of agents.

The first step is to present the implications of the data flag D. After setting D high, the agents are certain to possess certain knowledge about each other. Given the implications of the data flag, it will be shown that the reserved areas $T_{i,i,\mathbf{A}}$ and $T_{j,j,\mathbf{A}}$ never intersect. Therefore it is not possible for i and j to collide, since they have to remain within their respective reserved area $\mathbf{A}$. Note that the proof is symmetric with respect to i and j.

The proof that $T_{i,i,\mathbf{A}}$ and $T_{j,j,\mathbf{A}}$ never intersect is performed for time intervals $[t_0, t_e]$. Provided that initially at time t_0 there is no intersection, it will be shown that the agents can only adjust their reserved areas s.t. $T_{i,i,\mathbf{A}}$ and $T_{j,j,\mathbf{A}}$ stay distinct. The end of one time interval t_e is the beginning of the next interval. Hence, the reserved areas will never intersect if they were distinct when the algorithm started.

The time t_e is defined as the time when any of the following variables is being changed: $T_{i,j,D}$, $T_{j,i,D}$, $T_{i,j,R}$, $T_{j,i,R}$, $T_{i,j,\mathbf{A}}$, $T_{j,i,\mathbf{A}}$, $T_{i,j,\mathbf{B}}$, or $T_{j,i,\mathbf{B}}$. Note that changing $T_{i,i,\mathbf{B}}$ or $T_{j,j,\mathbf{B}}$ to an area that is not a subset of the previously reserved area causes an immediate change in $T_{i,j,D}$ ($T_{j,i,D}$ respectively) and ends the current time interval.

By definition of t_e, it follows that for t, s.t. $t_e \geq t > t_0$, the variables $T_{i,i,\mathbf{B}}$, $T_{j,j,\mathbf{B}}$, $T_{i,j,\mathbf{A}}$, $T_{i,j,\mathbf{B}}$, $T_{j,i,\mathbf{A}}$, $T_{j,i,\mathbf{B}}$, $T_{i,j,D}$, $T_{j,i,D}$, $T_{i,j,R}$, and $T_{j,i,R}$ are constant. Given the rules of changing

$T_{i,i,\mathbf{A}}$, (2.2) through (2.4), it follows that for $t_e \geq t > t_0$

$$\begin{aligned} T_{i,i,\mathbf{A}}(t) &\subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)]), \\ &\quad \text{if } (T_{i,j,D}(t_0) = 1) \wedge (T_{i,j,R}(t_0) = 1) \qquad \text{(C.1)} \\ T_{i,i,\mathbf{A}}(t) &\subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)]), \\ &\quad \text{if } (T_{i,j,D}(t_0) = 1) \wedge (T_{i,j,R}(t_0) = 0) \qquad \text{(C.2)} \\ T_{i,i,\mathbf{A}}(t) &\subseteq T_{i,i,\mathbf{A}}(t_0), \quad \text{otherwise} \qquad \text{(C.3)} \end{aligned}$$

Note that this holds true for times up to $t = t_e$, since a change of the information state at time t_e cannot affect **A** until a time $t > t_e$. This is due to the fact that $t(k_{i+1}) > t(k_i)$.

C.1 Unambiguity of priority R

One important property of the algorithm is that out of every pair of agents, only one can have the priority flag R at any given time. In the following it will be shown that the algorithm prevents both agents i and j from having priority at the same time. When an agent i makes the first contact with another agent j, $T_{i,j,R}$ and $T_{j,i,R}$ are being initialized to zero which means that initially no agent has priority, see pc:5c and pc:5d. It will be shown that if $T_{i,j,R}(t) = 1$ then $T_{j,i,R}(t) = 0$ and vice versa. For the following proofs assume without loss of generality that agent i is the PA.

C.1.1 Case 1: $T_{j,i,R}(t) = 1$

In order for $T_{j,i,R}(t) = 1$ to happen, j must have received a message P_1 at $t_{rp1} \leq t$ with $P_{1,R,\text{other}} = 1$, where $\bullet_{rp1}$ stands for "received message P_1". Denote the time when the message was sent by i as t_{sp1}, $t_{sp1} < t_{rp1}$. The flag $P_{1,R,\text{other}}$ is equal to $T_{i,j,R,\text{other}}(t_{sp1})$, see pc:9d. According to pc:7g and pc:7h, $T_{i,j,R,\text{other}}(t_{sp1}) = 1$ is only possible if $T_{i,j,R}(t_{sp1}) = 0$.

In order for i to gain priority as well it has to execute pc:7e, send another message P_2 to j and wait for the answer. However, the message P_2 will contain $P_{2,R,\text{other}} = 0$, a command to j to give up priority. Both P_1 and P_2 are messages from i to j and messages are sent/received strictly in order (see Section 2.2.3). Since $t_{sp1} < t_{sp2}$, j has to receive P_2 after P_1. As soon as j receives P_2 it sets $T_{j,i,R} = 0$, which violates the initial assumption of $T_{j,i,R}(t) = 1$. At the same time, i cannot gain priority before j receives P_2. This shows that $T_{i,j,R}(t) = 0$ if $T_{j,i,R}(t) = 1$.

C.1.2 Case 2: $T_{i,j,R}(t) = 1$

The flag $T_{i,j,R}(t) = 1$ can only be gained if $T_{i,j,R,\text{other}}(t) = 0$ and i receives a message P_1 from j at time $t_{rp1} \leq t$, with $P_{1,t,\text{other}} \geq T_{i,j,t5}$ and $P_{1,R} = 0$ (see pc:3j). Denote the time when P_1 was sent by j as t_{sp1}. From $P_{1,t,\text{other}} \geq T_{i,j,t5}$ it follows that j got a message P_2 from i previously, sent at $t_{sp2} \geq T_{i,j,t5}$. The sequence of events is therefore $T_{i,j,t5} \leq t_{sp2} \leq t_{sp1} \leq t$. The status of j at t_{sp1} must have been $T_{j,i,R}(t_{sp1}) = 0$, otherwise $P_{1,R}$ would not have been zero.

Now, for $T_{j,i,R} = 1$ to occur, j must have gained priority at some time between t_{sp1} and t. This is only possible if i sends another message P_3 at t_{sp3} with $T_{i,j,R,\text{other}} = 1$, which is received by j at t_{rp3}, such that $t_{sp1} < t_{rp3} \leq t$. Note that P_2 and P_3 are both messages from i to j. Since the order of messages is maintained (see Section 2.2.3) and $t_{rp3} > t_{rp2}$ it follows that $t_{sp3} > t_{sp2} > T_{i,j,t5}$. However, in order to set $T_{i,j,R,\text{other}} = 1$, i has to set $T_{i,j,R} = 0$ first, so $T_{i,j,R}(t_{sp2}) = 0$. In order to regain $T_{i,j,R} = 1$, i would have to execute pc:7e which would change $T_{i,j,t5}$ such that $T_{i,j,t5} > t_{sp2}$, leading to a contradiction. Therefore $T_{j,i,R}(t) = 0$ if $T_{i,j,R}(t) = 1$.

C.2 Implications of the data flag D and the priority flag R

The data flag D and the priority flag R are crucial to determine how the agents move. They have certain implications on the information state of the agents: If $T_{i,j,D}(t) = 1$, then there exists $(t_{i,2}, t_{i,3})$ with $t \geq t_{i,3} \geq t_{i,2} \geq t_{i,1}$, s.t.

$$T_{i,i,\mathbf{B}}(t) \subseteq T_{i,i,\mathbf{B}}(t_{i,2}) \subseteq T_{i,i,\mathbf{B}}(t_{i,1}) \tag{C.4}$$

$$T_{i,j,\mathbf{A}}(t) = T_{j,j,\mathbf{A}}(t_{i,2}) \tag{C.5}$$

$$T_{i,j,\mathbf{B}}(t) = T_{j,j,\mathbf{B}}(t_{i,2}) \tag{C.6}$$

$$T_{j,j,\mathbf{A}}(t) \subseteq T_{j,j,\mathbf{B}}(t_{i,2}) \tag{C.7}$$

$$T_{j,j,\mathbf{B}}(t) \subseteq T_{j,j,\mathbf{B}}(t_{i,2}), \quad \text{if } T_{j,i,D}(t) = 1 \tag{C.8}$$

If $T_{i,j,D}(t) = 1$ and $T_{i,j,R}(t) = 1$, then there exists $(t_{i,2}, t_{i,3})$ with $t \geq t_{i,3} \geq t_{i,2}$ and there exists $(t_{j,2}, t_{j,3})$ with $t \geq t_{j,3} \geq t_{j,2}$, s.t.

$$T_{j,j,\mathbf{A}}(t) \subseteq (T_{j,j,\mathbf{A}}(t_{i,2}) \cup [T_{j,j,\mathbf{B}}(t_{i,2}) \setminus T_{i,i,\mathbf{B}}(t_{j,2})]) \tag{C.9}$$

The statements (C.4) through (C.9) are proved by referring to the pseudo-code in Section 2.2.8.

Wireless transmission has a non-negative duration, hence $t_{i,3} \geq t_{i,2}$. As described in Section 2.2.3 a packet P_1 is only accepted if the time stamp $P_{1,t,\text{other}}$ is more recent than $t_{i,1}$, therefore $t_{i,2} \geq t_{i,1}$. It follows that the ordering of times is $t \geq t_{i,3} \geq t_{i,2} \geq t_{i,1}$. The same applies for j.

C.2.1 Proof of (C.4)

The fact that $T_{i,j,D}(t) = 1$ means that the handshake between i and j was completed and so there exists $(t_{i,3}, t_{i,2}, t_{i,1})$ the milestones of the handshake. By definition, $t_{i,1}$ is

the time when i sent the first packet of the handshake to j. This occurs at the end of the main loop. If i would have changed $T_{i,i,\mathbf{B}}$ s.t. the statement on line pc:5b would have been true, then $T_{i,j,D}$ would have been set to zero and the handshake would have had to start over, which is a contradiction. Hence, during every consecutive frame $T_{i,i,\mathbf{B}}$ had to be the subset of the previous $T_{i,i,\mathbf{B}}$. Since $t \geq t_{i,2} \geq t_{i,1}$ it follows that $T_{i,i,\mathbf{B}}(t) \subseteq T_{i,i,\mathbf{B}}(t_{i,2}) \subseteq T_{i,i,\mathbf{B}}(t_{i,1})$ which concludes the proof.

C.2.2 Proof of (C.5)

If $T_{i,j,D} = 1$ then pc:3g must have been executed. This is only possible as part of an information update. It follows that there exists $(t_{i,2}, t_{i,3})$ with $t \geq t_{i,3} \geq t_{i,2}$, since that is a requirement for a successful information update, see Section 2.2.3. The information state of i is being updated by setting $T_{i,j,\mathbf{A}} = P_{\text{in},\mathbf{A}}$, where $P_{\text{in},\mathbf{A}} = T_{j,j,\mathbf{A}}(t_{i,2})$ by definition of $t_{i,2}$. This concludes the proof.

C.2.3 Proof of (C.6)

This proof is analogous to the proof of (C.5) by simply substituting **B** for **A**.

C.2.4 Proof of (C.7)

Since $T_{j,i,D}(t) = 1$ there must have been at least one update from j to i. Denote the time when j sent the first update as $t^1_{i,2}$. During subsequent updates from j to i, $T_{j,j,\mathbf{B}}(t^{m+1}_{i,2}) \subseteq T_{j,j,\mathbf{B}}(t^m_{i,2})$ has to be true, otherwise $T_{i,j,D}$ would be set to zero (see 3d) and the initial assumption is violated. Also, $t^{m+1}_{i,2} > t^m_{i,2}$ since messages are only received in order. The final update is sent out at $t_{i,2}$, with the requested area being $T_{j,j,\mathbf{B}}(t_{i,2})$.

By definition of $\mathbf{B}$, $\mathbf{A} \subseteq \mathbf{B}$ for all t, therefore $T_{j,j,\mathbf{A}}(t_{i,2}) \subseteq T_{j,j,\mathbf{B}}(t_{i,2})$. So in order to change $T_{j,j,\mathbf{A}}$ to $T_{j,j,\mathbf{A}} \nsubseteq T_{j,j,\mathbf{B}}(t_{i,2})$, $T_{j,j,\mathbf{B}}$ has to be changed first. There must have been a time $t_{10} > t_{i,2}$ when $T_{j,j,\mathbf{B}}(t_{10}) \nsubseteq T_{j,j,\mathbf{B}}(t_{i,2})$. However, $\mathbf{A}$ can only be changed to something else than a subset of the previous $\mathbf{A}$ if the data flag $T_{j,i,D}$ is set high. The change from $T_{j,j,\mathbf{B}}(t_{i,2})$ to $T_{j,j,\mathbf{B}}(t_{10})$ would have pulled $T_{j,i,D}$ low. In order to regain $T_{j,i,D}$, j would have had to perform another handshake with i, which would have pulled $T_{i,j,D}$ low at some time t_{11}, $t \geq t_{11} \geq t_{10} > t_{i,2}$. This is a contradiction with the initial assumption of $T_{i,j,D}(t) = 1$. It follows that $T_{j,j,\mathbf{A}}(t) \nsubseteq T_{j,j,\mathbf{B}}(t_{i,2})$ is not possible which concludes the proof.

C.2.5 Proof of (C.8)

This proof follows along the same lines as the proof for (C.7). Assume that the last update from j to i was sent at time $t_{i,2}$. If j changes $T_{j,j,\mathbf{B}}$ at time $t_{12} > t_{i,2}$ such that $T_{j,j,\mathbf{B}}(t_{12}) \nsubseteq T_{j,j,\mathbf{B}}(t_{i,2})$ then $T_{j,i,D}(t_{12}) = 0$. Since j cannot regain $T_{j,i,D} = 0$ between $t_{i,2}$ and t (there is no more message from j to i) this leads to a contradiction with the initial assumption $T_{j,i,D}(t) = 1$. It follows that $T_{j,j,\mathbf{B}}(t) \subseteq T_{j,j,\mathbf{B}}(t_{i,2})$.

C.2.6 Proof of (C.9)

Following along the same lines as the proof for (C.7), define $t_{i,2}$ as the time when the last update from j to i was sent. Since only one agent can have R at any time it follows that $T_{j,i,R}(t') = 0$ for all t' such that $t \geq t' \geq t_{i,2}$. Also, in order to grant R to i, j must have had the data flag set high at some point, so that $t_{j,2}$ and $t_{j,3}$ exist.

After giving up priority, j can still adjust $T_{j,j,\mathbf{A}}$ according to (2.3) or (2.4), depending on whether j's data flag is set high. The more general case is $T_{j,j,D}(t') = 1$, since the

$T_{j,j,\mathbf{A}}$ in (2.3) is a superset of $T_{j,j,\mathbf{A}}$ in (2.4). From now on assume $T_{j,j,D}(t') = 1$. Using (2.3) it follows that

$$T_{j,j,\mathbf{A}}(t') \quad \subseteq \quad (T_{j,j,\mathbf{A}}(t_{i,2}) \cup [T_{j,j,\mathbf{B}}(t_{i,2}) \setminus T_{j,i,\mathbf{B}}(t')]) \tag{C.10}$$

Since $T_{j,i,D}(t') = 1$, j must have received an information update from i. By definition, $T_{i,i,\mathbf{B}}(t_{j,2})$ with $t \geq t' \geq t_{j,2}$ is the latest information j has about i, therefore

$$T_{j,j,\mathbf{A}}(t') \quad \subseteq \quad (T_{j,j,\mathbf{A}}(t_{i,2}) \cup [T_{j,j,\mathbf{B}}(t_{i,2}) \setminus T_{i,i,\mathbf{B}}(t_{j,2})]) \tag{C.11}$$

which concludes the proof.

C.3 No intersection between reserved areas A

In this section it will be shown that the algorithm keeps the areas $T_{i,i,\mathbf{A}}$ and $T_{j,j,\mathbf{A}}$ distinct until time t_e

$$T_{i,i,\mathbf{A}}(t) \cap T_{j,j,\mathbf{A}}(t) \quad = \quad \emptyset, \quad t_\mathrm{e} \geq t > t_0 \tag{C.12}$$

provided that they do not intersect initially

$$T_{i,i,\mathbf{A}}(t) \cap T_{j,j,\mathbf{A}}(t) \quad = \quad \emptyset, \quad t \leq t_0 \tag{C.13}$$

Depending on the data and priority flags, agents i and j have different options of how to adjust their reserved areas. Table C.1 shows all possible combinations. Cases like ($T_{i,j,D} = 0$, $T_{j,i,D} = 1$) have been dropped without loss of generality, since the proof is symmetric in i and j. Also, it is not possible for both agents to have the R flag at the same time. It will be shown that $T_{i,i,\mathbf{A}}$ and $T_{j,j,\mathbf{A}}$ remain distinct in all 5 possible cases.

Table C.1: Cases to prove

Case	$T_{i,j,D}$	$T_{j,i,D}$	$T_{i,j,R}$	$T_{j,i,R}$
1	0	0	0	0
2	1	0	0	0
3	1	0	1	0
4	1	1	0	0
5	1	1	1	0

C.3.1 Case 1

With $T_{i,j,D}(t) = 0$ and $T_{j,i,D}(t) = 0$ for $t_\mathrm{e} > t \geq t_0$ it follows from (C.3) that for $t_\mathrm{e} \geq t' > t_0$

$$T_{i,i,\mathbf{A}}(t') \subseteq T_{i,i,\mathbf{A}}(t_0) \tag{C.14}$$

$$T_{j,j,\mathbf{A}}(t') \subseteq T_{j,j,\mathbf{A}}(t_0) \tag{C.15}$$

Using the initial condition (C.13) yields in a straightforward manner

$$T_{i,i,\mathbf{A}}(t') \cap T_{j,j,\mathbf{A}}(t') = \emptyset, \quad t_\mathrm{e} \geq t' > t_0 \tag{C.16}$$

and the areas remain distinct.

C.3.2 Case 2

With $T_{i,j,D}(t) = 1$, $T_{i,j,R}(t) = 0$, and $T_{j,i,D}(t) = 0$ for $t_\mathrm{e} > t \geq t_0$ it follows from (C.2) and (C.3) that for $t_\mathrm{e} \geq t' > t_0$

$$T_{i,i,\mathbf{A}}(t') \subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)]) \tag{C.17}$$

$$T_{j,j,\mathbf{A}}(t') \subseteq T_{j,j,\mathbf{A}}(t_0) \tag{C.18}$$

In order to prove that $T_{i,i,\mathbf{A}}(t')$ and $T_{j,j,\mathbf{A}}(t')$ are distinct it is sufficient to show that the following two equations hold true:

$$T_{j,j,\mathbf{A}}(t_0) \cap T_{i,i,\mathbf{A}}(t_0) = \emptyset \tag{C.19}$$

$$T_{j,j,\mathbf{A}}(t_0) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)] = \emptyset \tag{C.20}$$

Equation (C.19) is true by the initial condition (C.13). The fact that $T_{i,j,D}$ is equal to 1 implies (C.6) and (C.7). Substituting t_0 for t in (C.6) and (C.7) yields $T_{i,j,\mathbf{B}}(t_0) = T_{j,j,\mathbf{B}}(t_{i,2}) \supseteq T_{j,j,\mathbf{A}}(t_0)$. Therefore the term on the right hand side of the intersection of (C.20) explicitly excludes $T_{j,j,\mathbf{A}}(t_0)$ and (C.20) holds true. This shows that

$$T_{i,i,\mathbf{A}}(t') \cap T_{j,j,\mathbf{A}}(t') = \emptyset, \quad t_\mathrm{e} \geq t' > t_0 \tag{C.21}$$

C.3.3 Case 3

With $T_{i,j,D}(t) = 1$, $T_{i,j,R}(t) = 1$, and $T_{j,i,D}(t) = 0$ for $t_\mathrm{e} > t \geq t_0$ it follows from (C.1) and (C.3) that for $t_\mathrm{e} \geq t' > t_0$

$$T_{i,i,\mathbf{A}}(t') \subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)]) \tag{C.22}$$

$$T_{j,j,\mathbf{A}}(t') \subseteq T_{j,j,\mathbf{A}}(t_0) \tag{C.23}$$

In order to prove that $T_{i,i,\mathbf{A}}(t')$ and $T_{j,j,\mathbf{A}}(t')$ are distinct it is sufficient to show that the following two equations hold true:

$$T_{j,j,\mathbf{A}}(t_0) \cap T_{i,i,\mathbf{A}}(t_0) = \emptyset \tag{C.24}$$

$$T_{j,j,\mathbf{A}}(t_0) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)] = \emptyset \tag{C.25}$$

Equation (C.24) is true by the initial condition (C.13). Substitute t_0 for t in (C.9) and plug the result into (C.25):

$$(T_{j,j,\mathbf{A}}(t_{i,2}) \cup [T_{j,j,\mathbf{B}}(t_{i,2}) \setminus T_{i,i,\mathbf{B}}(t_{j,2})]) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)] = \emptyset \tag{C.26}$$

In order to show that (C.26) is true, use (C.5) and break it up into two equations:

$$T_{j,j,\mathbf{A}}(t_{i,2}) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{j,j,\mathbf{A}}(t_{i,2})] = \emptyset \tag{C.27}$$

$$(T_{j,j,\mathbf{B}}(t_{i,2}) \setminus T_{i,i,\mathbf{B}}(t_{j,2})) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{j,j,\mathbf{A}}(t_{i,2})] = \emptyset \tag{C.28}$$

Equation (C.27) is trivially true. Note that with (C.4), $T_{i,i,\mathbf{B}}(t_{j,2}) \supseteq T_{i,i,\mathbf{B}}(t_0)$ (since $t_{j,2} \geq t_0$) and the term on the left hand side of the intersection of (C.28) explicitly excludes $T_{i,i,\mathbf{B}}(t_0)$, which shows that

$$T_{i,i,\mathbf{A}}(t') \cap T_{j,j,\mathbf{A}}(t') = \emptyset, \quad t_\mathrm{e} \geq t' > t_0 \tag{C.29}$$

C.3.4 Case 4

With $T_{i,j,D}(t) = 1$, $T_{i,j,R}(t) = 0$, $T_{j,i,D}(t) = 1$, and $T_{j,i,R}(t) = 0$ for $t_\mathrm{e} > t \geq t_0$ it follows from (C.2) that for $t_\mathrm{e} \geq t' > t_0$

$$T_{i,i,\mathbf{A}}(t') \subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)]) \tag{C.30}$$

$$T_{j,j,\mathbf{A}}(t') \subseteq (T_{j,j,\mathbf{A}}(t_0) \cup [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)]) \tag{C.31}$$

In order to prove that $T_{i,i,\mathbf{A}}(t')$ and $T_{j,j,\mathbf{A}}(t')$ are distinct it is sufficient to show that the following four equations hold true:

$$T_{j,j,\mathbf{A}}(t_0) \cap T_{i,i,\mathbf{A}}(t_0) = \emptyset \tag{C.32}$$

$$T_{j,j,\mathbf{A}}(t_0) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)] = \emptyset \tag{C.33}$$

$$T_{i,i,\mathbf{A}}(t_0) \cap [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)] = \emptyset \tag{C.34}$$

$$(T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{B}}(t_0)) \cap [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)] = \emptyset \tag{C.35}$$

Equation (C.32) is true by the initial condition (C.13). Equations (C.33) and (C.34) can be proved in the same way as case 2, which yields $T_{i,j,\mathbf{B}}(t_0) = T_{j,j,\mathbf{B}}(t_{i,2}) \supseteq T_{j,j,\mathbf{A}}(t_0)$ and $T_{j,i,\mathbf{B}}(t_0) = T_{i,i,\mathbf{B}}(t_{j,2}) \supseteq T_{i,i,\mathbf{A}}(t_0)$. Equation (C.35) is proved by substituting t_0 for t in

(C.6) and (C.8), which yields $T_{i,j,\mathbf{B}}(t_0) = T_{j,j,\mathbf{B}}(t_{i,2}) \supseteq T_{j,j,\mathbf{B}}(t_0)$. Therefore the term on the left hand side of the intersection of (C.35) explicitly excludes $T_{j,j,\mathbf{B}}(t_0)$ and (C.35) holds true. This shows that

$$T_{i,i,\mathbf{A}}(t') \cap T_{j,j,\mathbf{A}}(t') = \emptyset, \quad t_\mathrm{e} \geq t' > t_0 \tag{C.36}$$

C.3.5 Case 5

With $T_{i,j,D}(t) = 1$, $T_{i,j,R}(t) = 1$, $T_{j,i,D}(t) = 1$, and $T_{j,i,R}(t) = 0$ for $t_\mathrm{e} > t \geq t_0$ it follows from (C.1) and (C.2) that for $t_\mathrm{e} \geq t' > t_0$

$$T_{i,i,\mathbf{A}}(t') \subseteq (T_{i,i,\mathbf{A}}(t_0) \cup [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)]) \tag{C.37}$$

$$T_{j,j,\mathbf{A}}(t') \subseteq (T_{j,j,\mathbf{A}}(t_0) \cup [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)]) \tag{C.38}$$

In order to prove that $T_{i,i,\mathbf{A}}(t')$ and $T_{j,j,\mathbf{A}}(t')$ are distinct it is sufficient to show that the following four equations hold true:

$$T_{j,j,\mathbf{A}}(t_0) \cap T_{i,i,\mathbf{A}}(t_0) = \emptyset \tag{C.39}$$

$$T_{j,j,\mathbf{A}}(t_0) \cap [T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)] = \emptyset \tag{C.40}$$

$$T_{i,i,\mathbf{A}}(t_0) \cap [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)] = \emptyset \tag{C.41}$$

$$(T_{i,i,\mathbf{B}}(t_0) \setminus T_{i,j,\mathbf{A}}(t_0)) \cap [T_{j,j,\mathbf{B}}(t_0) \setminus T_{j,i,\mathbf{B}}(t_0)] = \emptyset \tag{C.42}$$

Equation (C.39) is true by the initial condition (C.13). Equations (C.41) and (C.42) can be proved in the same way as done in case 4. Equation (C.40) can be proved in the same way as done in case 3, i.e., substituting t_0 for t in (C.9) and plugging the result into (C.42), then breaking it up into two separate equations, proving them separately. This shows that

$$T_{i,i,\mathbf{A}}(t') \cap T_{j,j,\mathbf{A}}(t') = \emptyset, \quad t_\mathrm{e} \geq t' > t_0 \tag{C.43}$$

C.4 No collisions

It has been shown that

$$T_{i,i,\mathbf{A}}(t) \cap T_{j,j,\mathbf{A}}(t) = \emptyset, \quad \forall t \tag{C.44}$$

if both agents adhere to the algorithm. According to (2.5) an agent i can only move such that its path will keep it inside $T_{i,i,\mathbf{A}}$. A collision between two agents occurs if there exists t_{coll} s.t.

$$\mathbf{G}(\mathbf{p}_i(t_{\text{coll}})) \cap \mathbf{G}(\mathbf{p}_j(t_{\text{coll}})) \neq \emptyset \tag{C.45}$$

From (C.44) and (2.5) it follows that for all t

$$\mathbf{G}(\mathbf{p}_i(t)) \cap \mathbf{G}(\mathbf{p}_j(t)) = \emptyset \tag{C.46}$$

and the proof of safety is complete.

APPENDIX D
DESCRIPTION OF THE AUTONOMOUS FLYING VEHICLE (AFV)

The vehicle was developed and built at Cornell University [60]. The following section presents aspects of the design which complement the algorithm and experimentation covered above.

D.1 Mechanical design

The four propellers of the quadrotor UAV are mounted at the corners of a wire-frame structure. This provides significant stiffness and rigidity while keeping the vehicle light. The structure consists of a series of struts extending from the vehicle's center to each motor mount. Four stiffening wires are affixed to the end of each strut. The propellers have fixed pitch and are driven by four geared brushless motors. The produced thrust can be changed by adjusting the rpm of the propellers. Analog power is provided by eight packs of lithium-polymer batteries with a total capacity of 20.8 Ah. The onboard electronics and sensor unit package is mounted at the center of the structure. Table D.1 shows the mechanical parameters of the vehicle.

Table D.1: Mechanical Parameters of the UAV

Dimensions	Propeller ∅	Max vert. acc.	Motor rated power
1.5 x 1.5 x 0.4 m^3	0.465 m	1 g beyond hover	1200 W (per motor)

D.2 Electrical design

The electrical system is designed to read the sensor data, control the actuators, and send telemetry data back to the base station, see Figure D.1. A Pentium class single board computer running Windows CE processes data for state estimation and the main controller. This processor communicates with all sensors and the motor control units via an interface board. Given the data from the inertial measurement unit (IMU) and position data from the Vicon camera system, it computes desired propeller speeds. These are sent to each motor control unit. The motor control units are independent systems, which perform local PI feedback control and provide power and commutation for the brushless motors. For data downlink and command upload the vehicle communicates with the ground station by utilizing an 802.11a wireless interface.

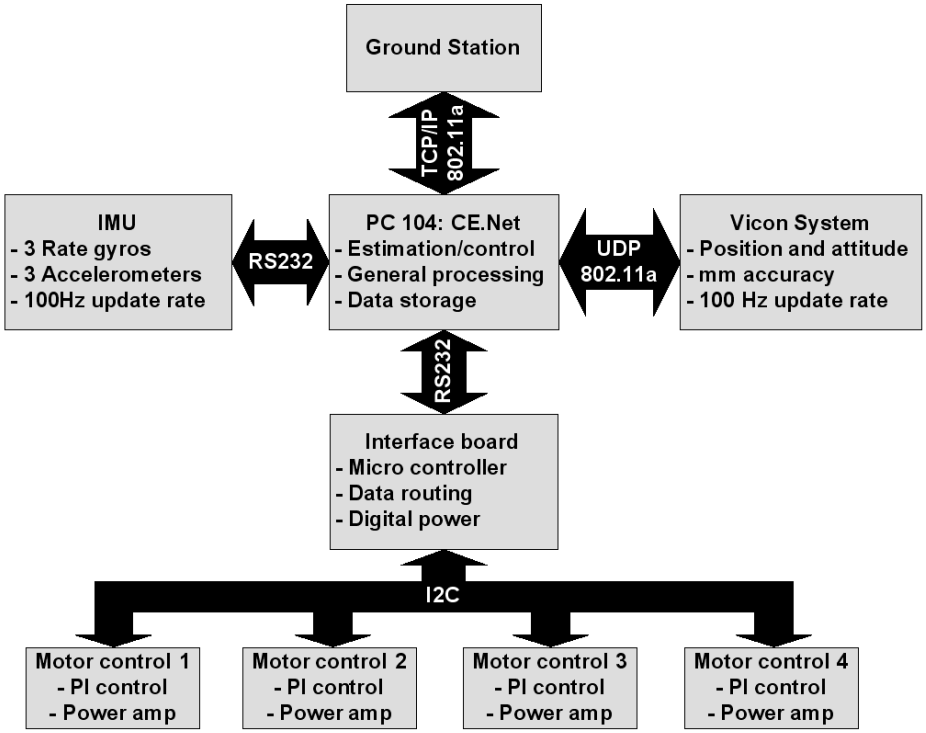

Figure D.1: The Electrical System

D.3 Sensors

The IMU is the primary navigation sensor of the vehicle, providing inertial data at an update rate of 100 Hz [62]. The IMU is composed of three orthogonal accelerometers and gyroscopes which deliver measurements of acceleration and rotation rates in all three dimensions. The Vicon camera system provides absolute position and attitude data in all three dimensions [63]. The measurement error of the Vicon system is on the order of one millimeter. Featuring an update rate of 100 Hz it is used as part of the extended Kalman Filter to compensate for the drift of the IMU measurements.

D.4 State estimation

The state estimator is an extended (nonlinear) Kalman Filter which fuses the measurements from the IMU and Vicon system. The implementation uses variable state covariance, i.e. the state covariance has to be propagated along with the state estimate. This standard approach is described in the literature, for example [39], hence the following paragraphs will be limited to the definition of the inputs, outputs, and dynamics. The state $q_{3d,est}$ is defined as

$$q_{3d,est} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \phi & \theta & \psi & \omega_n & \omega_o & \omega_a \end{bmatrix}^T \tag{D.1}$$

with $E = [\phi, \theta, \psi]^T$ denoting the attitude of the vehicle in Euler angles, and $\omega_b = [\omega_n, \omega_o, \omega_a]^T$ denoting angle rates in the body coordinate system *noa*. Note that the indices $3d$ indicate dimensions and forces of the 3D vehicle, not of the equivalent 2D version. The IMU measures the angular rates ω_b and the acceleration $\ddot{p}_b$ of the vehicle in body coordinates. These measurements are treated as inputs for the state propagation step of the estimator. It is assumed that the measurements are corrupted by constant

offsets $b_\bullet$ and iid Gaussian white noise $v_\bullet$

$$\omega_{b,m} = \omega_b + b_\omega + v_\omega, \quad E[v_\omega] = 0, \quad E[v_\omega v_\omega^T] = V_\omega \tag{D.2}$$

$$\ddot{p}_{b,m} = \ddot{p}_b + b_{acc} + v_{acc}, \quad E[v_{acc}] = 0, \quad E[v_{acc} v_{acc}^T] = V_{acc} \tag{D.3}$$

with $\omega_{b,m}$ and $\ddot{p}_{b,m}$ denoting the noisy measurements of the rate gyros and accelerometers respectively. The constant offsets are determined before taking off when the vehicle is motionless on the landing platform. The nonlinear estimator dynamics are defined as

$$\dot{q}_{3d,est} = \begin{bmatrix} \dot{p}_{est} \\ +g_i + A^T(E)[\ddot{p}_{b,m} - b_{acc} - v_{acc}] \\ M^{-1}(E)A^T(E)[\omega_{b,m} - b_\omega - v_\omega] \\ \dot{\omega}_m \end{bmatrix} \tag{D.4}$$

$$h_{3d,est} = \begin{bmatrix} p + v_p \\ E + v_E \end{bmatrix} \tag{D.5}$$

where p_{est} is the estimate of the position vector $p = [xyz]$, g_i is the gravity vector in inertial coordinates xyz, $h_{3d,est}$ defines the measurement update (measurements by the Vicon system), v_p and v_E are the Vicon measurement noise, $A(E)$ is the transformation from inertial to body coordinates, and $M^{-1}(E)$ is the transformation from global rates to Euler rates:

$$A(E) = \begin{bmatrix} \mathrm{c}(\psi)\mathrm{c}(\theta) & \mathrm{s}(\psi)\mathrm{c}(\theta) & -\mathrm{s}(\theta) \\ \mathrm{c}(\psi)\mathrm{s}(\theta)\mathrm{s}(\phi) - \mathrm{s}(\psi)\mathrm{c}(\phi) & \mathrm{s}(\psi)\mathrm{s}(\theta)\mathrm{s}(\phi) + \mathrm{c}(\psi)\mathrm{c}(\phi) & \mathrm{c}(\theta)\mathrm{s}(\phi) \\ \mathrm{c}(\psi)\mathrm{s}(\theta)\mathrm{c}(\phi) + \mathrm{s}(\psi)\mathrm{s}(\phi) & \mathrm{s}(\psi)\mathrm{s}(\theta)\mathrm{c}(\phi) - \mathrm{c}(\psi)\mathrm{s}(\phi) & \mathrm{c}(\theta)\mathrm{c}(\phi) \end{bmatrix} \tag{D.6}$$

$$M^{-1}(E) = \begin{bmatrix} \cos(\psi)/\cos(\theta) & \sin(\psi)/\cos(\theta) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ \cos(\psi)\sin(\theta)/\cos(\theta) & \sin(\psi)\sin(\theta)/\cos(\theta) & 1 \end{bmatrix} \tag{D.7}$$

where cos() and sin() have been abbreviated by c() and s().

Note that in practice the fourth line of (D.4) is not being propagated through. Instead, the estimate of the vehicles angular rate is made equal to the most recent measurement minus the bias $\omega_{b,m} - b_\omega$.

D.5 Controller

The goal of the controller is to stabilize the UAV in hover ($\dot{q}_{3d} = 0$), given an estimate of the state vector (D.1). The controller is designed such that the vehicle response is that of a 2nd order system

$$\ddot{E} = -\gamma_E(E - E_d) - \beta_E \dot{E} \tag{D.8}$$

$$\ddot{p} = -\gamma_p(p - p_d) - \beta_p \dot{p} \tag{D.9}$$

where γ and β are the controller parameters and the index d denotes the desired angle or position. The six DOF rigid body dynamics of the vehicle are

$$\ddot{p} = A^T(E) \begin{bmatrix} 0 \\ 0 \\ -f_{3d,a} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{D.10}$$

$$j_n \dot{\omega}_n = \tau_n - (j_a - j_o)\omega_a \omega_o \tag{D.11}$$

$$j_o \dot{\omega}_o = \tau_o - (j_n - j_a)\omega_n \omega_a \tag{D.12}$$

$$j_a \dot{\omega}_a = \tau_a - (j_o - j_n)\omega_o \omega_n \tag{D.13}$$

with $f_{3d,a} = \sum_i f_{3d,i}$ denoting the common thrust vector, $\tau_\bullet$ denoting torques acting on the vehicle, and $j_\bullet$ indicating the moments of inertia.

Due to the propeller arrangement the lateral vehicle motion cannot be controlled directly. Instead, rotation about ϕ or θ results in a tilt of the thrust vector, which causes acceleration in x and y. For small changes $\Delta\theta$ and $\Delta\phi$ about hover it can readily be

shown from (D.10) that

$$\Delta\ddot{x} = -mg\Delta\theta \tag{D.14}$$

$$\Delta\ddot{y} = mg\Delta\phi \tag{D.15}$$

By using (D.14) and (D.15) the translational feedback in x or y (D.9) can be expressed in terms of rotations

$$E_{d,xy} = \begin{bmatrix} \phi_{d,y} \\ \theta_{d,x} \end{bmatrix} = \begin{bmatrix} 0 & mg & 0 \\ -mg & 0 & 0 \end{bmatrix} \left[-\gamma_p(p - p_d) - \beta_p \dot{p} \right] \tag{D.16}$$

which yields the following feedback laws

$$\ddot{E} = -\gamma_E(E - E_d - E_{d,xy}) - \beta_E \dot{E} \tag{D.17}$$

$$\ddot{z} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \left[-\gamma_p(p - p_d) - \beta_p \dot{p} \right] \tag{D.18}$$

The transformations from body rates to Euler angles and their derivatives are

$$\dot{E} = M^{-1}(E)A^T(E)\omega_b \tag{D.19}$$

$$\begin{aligned} \ddot{E} = {} & M^{-1}(E)A^T(E)\dot{\omega}_b \\ & + \left[\frac{\partial(M^{-1}A)}{\partial\phi}\dot{\phi} + \frac{\partial(M^{-1}A)}{\partial\theta}\dot{\theta} + \frac{\partial(M^{-1}A)}{\partial\psi}\dot{\psi} \right] \omega_b \end{aligned} \tag{D.20}$$

The relationships between torques/forces and individual thrust forces $f_\bullet$ created by the propellers are

$$\begin{bmatrix} \tau_n \\ \tau_o \\ \tau_a \\ f_{3d,a} \end{bmatrix} = \begin{bmatrix} 0 & -l_{3d,1} & 0 & l_{3d,3} \\ l_{3d,0} & 0 & -l_{3d,2} & 0 \\ -k_{FD} & k_{FD} & -k_{FD} & k_{FD} \\ -1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} f_{3d,0} \\ f_{3d,1} \\ f_{3d,2} \\ f_{3d,3} \end{bmatrix} \tag{D.21}$$

with $k_{FD} = k_{Drag}/k_{Thrust}$, and $l_{3d,i}$ denoting the moment arm of $f_{3d,i}$ w.r.t. the center of mass.

The final controller can be computed by substituting (D.10) through (D.13), (D.19), (D.20), and (D.21) into (D.17) and (D.18). Linearizing about hover and replacing E, $\dot{E}$, p, and $\dot{p}$ with the respective estimates from the state estimator yields the matrix K s.t.

$$f_{3d} = Kq_{3d,est}, \quad f_{3d} = \begin{bmatrix} f_{3d,0} & f_{3d,1} & f_{3d,2} & f_{3d,3} \end{bmatrix}^T \tag{D.22}$$

Given the desired thrust $f_{3d,i}$ an ideal angular velocity ω_i for each propeller i is commanded to the local PI motor control loops. It is assumed that the relationships between $f_{3d,i}$ and ω_i can be modeled [57] as

$$f_i = k_{Thrust}\omega_i^2 \tag{D.23}$$

$$d_i = k_{Drag}\omega_i^2 \tag{D.24}$$

which accounts for some of the nonlinearity in the propeller aerodynamics around the hover state. The parameters k_{Thrust} and k_{Drag} have been determined experimentally [60].

BIBLIOGRAPHY

[1] Gillespie T.D.: "Fundamentals of Vehicle Dynamics", Society of Automotive Engineers, 1992

[2] Bryson A.E., Ho Y.-C.: "Applied Optimal Control", John Wiley & Sons, 1975

[3] Moon, Francis C.: "Applied Dynamics", Wiley-Interscience, 1998

[4] Strichartz R.S.: "The Way of Analysis", Jones and Bartlett, 2000

[5] Kalmár-Nagy T., D'Andrea R., Ganguly P.: "Near-Optimal Dynamic Trajectory Generation and Control of an Omnidirectional Vehicle", Robotics and Autonomous Systems, Vol. 46, 2004, pp. 47-64

[6] Purwin O., D'Andrea R.: "Cornell Big Red 2003", in: Polani D., Bonarini A., Browning B., Yoshida K. (Eds), Robocup 2003: Robot Soccer World Cup VII, Lecture Notes in Artificial Intelligence, Springer, Berlin, 2003

[7] D'Andrea R., Kalmár-Nagy T., Ganguly P., Babish M.: "The Cornell RoboCup Team", in: Stone P., Balch T., Kraetzschmar (Eds), Robocup 2000: Robot Soccer World Cup IV, Springer, Berlin, 2001

[8] Moore K.L., Flann N.S.: "A Six-Wheeled Omnidirectional Autonomous Mobile Robot", Control Systems Magazine, IEEE, Vol. 20, Issue 6, Dec 2000, pp. 53-66

[9] Watanabe K., Shiraishi Y., Tzafestas S.G., Tang J., Fukuda T.: "Feedback Control of an Omnidirectional Autonomous Platform for Mobile Service Robots", Journal of Intelligent and Robotic Systems, Vol. 22, Issue 3-4, 1998, pp. 315-330

[10] Frazzoli E., Dahleh M.A., Feron E.: "Real-Time Motion Planning for Agile Autonomous Vehicles", Journal of Guidance, Control, and Dynamics, Vol. 25, No. 1, January 2002, pp. 116-129(14)

[11] Paromtchik I.E., Rembold U.: "A practical approach to motion generation and control for an omnidirectional mobile robot", Proceedings of IEEE International Conference on Robotics and Automation, Vol. 4, 1994, pp. 2790-2795

[12] Muñoz V., Ollero A., Prado M., Simón A.: "Mobile Robot Trajectory Planning with Dynamic and Kinematic Constraints", Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 4, 1994, pp. 2802-2807

[13] Faiz N., Agrawal S.K.: "Trajectory Planning of Robots with Dynamics and Inequalities", Proceedings of the 2000 IEEE International Conference on Robotics and Automation, Vol. 4, 2000, pp. 3976-3982

[14] Liu Y., Wu X., Zhu J., Lew J.: "Omni-directional mobile robot controller design by trajectory linearization", Proceedings of the American Control Conference 2003, Vol. 4, No. 4-6, June 2003, pp. 3423-3428

[15] Olsson H., Aström K.J., Canudas de Wit C., Gäfvert M., Lischinsky P.: "Friction Models and Friction Compensation"

[16] Bender C.B., Brody D.C., Meister B.K.: "Quantised Three Pillar Problem", submitted to Journal of Physics A, 2002

[17] Plumlee, J.: "Multi-Input Ground Vehicle Control Using Quadratic Programming Based Control Allocation Techniques", M.S. thesis, Auburn University, Auburn, Alabama, August 2004

[18] Purwin, O., D'Andrea R.: "Continuity and Monotonicity Properties of an Optimally Controlled Double Integrator with State and Input Constraints", TR2005-2001, Cornell University Library Technical Reports and Papers, 2005, http://techreports.library.cornell.edu

[19] Cormen T.H., Leiserson C.E., Rivest R.L., Stein C.: "Introduction to Algorithms", McGraw-Hill, 2001

[20] Kuchar J.K., Yang L.C.: "A Review of Conflict Detection and Resolution Modeling Methods", IEEE Transactions on Intelligent Transportation Systems, Special Issue on Air Traffic Control Part I, Vol. 1, No. 4, 2000, pp. 179-189

[21] Tomlin C., Pappas G., Sastry S.: "Conflict Resolution for Air Traffic Management: a Study in Multi-Agent Hybrid Systems", IEEE Transactions on Automatic Control , Vol. 43, No. 4, 1998, pp. 509-521

[22] Hu J., Pradini M., Sastry S.: "Three Dimensional Optimal Coordinated Maneuvers for Aircraft Conflict Avoidance", Journal of Guidance, Control, and Dynamics, Vol. 25, No. 5, 2002, pp. 888-900

[23] Frazzoli E., Mao Z.-H., Oh J.-H., Feron E.: "Resolution of Conflicts Involving Many Aircraft via Semidefinite Programming", AIAA Journal of Guidance, Control and Dynamics, Vol. 24, No. 1, 2001, pp. 79-86

[24] Raghunathan A., Gopal V., Subramanian D., Biegler L.T., Samad T.: "Dynamic Optimization Strategies for 3D Conflict Resolution of Multiple Aircraft", AIAA Journal of Guidance, Control and Dynamics, 27 (4), pp. 586-594 (2004)

[25] Pallottino L., Scordio V. G., Frazzoli E., and Bicchi A.: "Probabilistic verification of a decentralized policy for conflict resolution in multi-agent systems", International Conf. on Robotics and Automation, Orlando, FL, pp. 2448-2453, 2006

[26] Schouwenaars T., How J.P., Feron E.: "Decentralized Cooperative Trajectory Planning of Multiple Aircraft with Hard Safety Guarantees", Proceedings of the AIAA Guidance, Navigation and Control Conference, Aug. 2004. AIAA-2004-5141

[27] Inalhan G., Stipanovic D., Tomlin C.: "Decentralized Optimization, with Application to Multiple Aircraft Coordination", Proc. 41st IEEE Conference on Decision and Control, Las Vegas, NV, Dec. 2002

[28] Hwang I., Tomlin C.J.: "Protocol-Based Conflict Resolution for Air Traffic Control", Technical Rept., Dept. of Aeronautics and Astronautics, Stanford Univ., Stanford, CA, 2001

[29] Sigurd K., How J.P.: "UAV Trajectory Design Using Total Field Collision Avoidance", Proceedings of the AIAA Guidance, Navigation, and Control Conference, August 2003. AIAA-2003-5728

[30] Shim D.H., Kim H.J., Sastry S.: "Decentralized Nonlinear Model Predictive Control of Multiple Flying Robots in Dynamic Environments", Proc. 44th IEEE Conference on Decision and Control (CDC03), Maui, HI, Dec. 2003

[31] Schneider D., Campbell M.: "Real Time Optimal Task Allocation in Highly Dynamic Environments", ASME International Mechanical Engineering Congress and Exposition, Orlando, FL, November 2005

[32] Elston J., Frew E.W., Argrow B.:"Networked UAV Communication, Command, and Control", AIAA Guidance, Navigation, and Control Conference, Keystone, CO, August 2006

[33] Breger L., How J.P.: "Safe Trajectories for Autonomous Rendezvous of Spacecraft", AIAA Guidance, Navigation, and Control Conference, Keystone, CO, August 2006

[34] Chiang Y.-J., Klosowski J.T., Lee C., Mitchell J.S.B.: "Geometric Algorithms for Conflict Detection/Resolution in Air Traffic Management", Proceedings of the

36th IEEE Conference on Decision and Control, Vol. 2, IEEE Publ., Piscataway, NJ, 1997, pp. 1835-1840

[35] Purwin O., D'Andrea R.: "Trajectory generation and control for four wheeled omnidirectional vehicles", Robotics and Autonomous Systems, Volume 54, Issue 1, 2006, Pages 13-22, DOI: 10.1016/j.robot.2005.10.002, http://authors.elsevier.com/sd/article/S0921889005001673

[36] Haines E.: "Point in Polygon Strategies", Graphics Gems IV, ed. Paul Heckbert, Academic Press, p. 24-46, 1994

[37] Sherback M., Purwin O., D'Andrea R.: "Real-Time Motion Planning and Control in the 2005 Cornell RoboCup System", Lecture Notes in Control and Information Sciences, Volume 335, 2006, Pages 245-263, DOI: 10.1007/11397540

[38] Boyd S., Vandenberghe L.: "Convex Optimization", Cambridge University Press, 2004, ISBN: 978-0521833783

[39] Bar-Shalom Y., Li X.R., Kirubarajan T.: "Estimation with Applications to Tracking and Navigation", Wiley-Interscience, 2001, ISBN: 978-0471416555

[40] Bristow D.A., Tharayil M., Alleyne A.G.: "A survey of iterative learning control", IEEE Control Systems Magazine, Vol. 26, No. 3, pp. 96-114, 2006

[41] Chen Y.Q., Moore K.L.: "A Practical Iterative Learning Path-Following Control of an Omni-Directional Vehicle", Special Issue on Iterative Learning Control, Asian Journal of Control, Vol. 4, No. 1, pp. 90-98, 2002

[42] Schaal S., Atkeson C.G.: "Robot Juggling: An Implementation of Memory-based Learning", Control Systems Magazine, Vol. 14, No. 1, pp. 57-71, 1994

[43] Hjalmarsson H.: "Iterative feedback tuning - an overview", International Journal of Adaptive Control and Signal Processing, Vol. 16, pp. 373-395, 2002, DOI: 10.1002/acs.714

[44] Jansson H., Hjalmarsson H.: "Gradient approximations in iterative feedback tuning for multivariable processes", International Journal of Adaptive Control and Signal Processing, Vol. 18, pp. 665-681, 2004, DOI: 10.1002/acs.826)

[45] Campi M.C., Lecchini A., Savaresi S.M.: "Virtual reference feedback tuning: a direct method for the design of feedback controllers", Automatica, Vol. 38, pp. 1337-1346, 2002

[46] Lecchini A., Campi M.C., Savaresi S.M.: "Virtual Reference Feedback Tuning for Two Degree of Freedom Controllers", International Journal of Adaptive Control and Signal Processing, Vol. 16, pp. 355-371, 2002

[47] Abbeel P., Coates A., Quigley M., Ng A.Y.:"An Application of Reinforcement Learning to Aerobatic Helicopter Flight", NIPS, Vol. 19, 2007

[48] Abbeel P.: "Apprenticeship Learning and Reinforcement Learning with Application to Robotic Control", Ph.D. Dissertation, Stanford University, Computer Science, August 2008

[49] Lee K.S., Lee J.H., Chin I.S., Lee H.J.: "A model predictive control technique for batch processes and its application to temperature tracking control of an experimental batchreactor", A.I.Ch.E. Journal, Vol. 45, No. 10, pp. 2175-2187, 1999

[50] Lee J.H., Lee K.S., Kim W.C.: "Model-based iterative learning control with a quadratic criterion for time-varying linear systems", Automatica, Vol. 36, pp. 641-657, 2000

[51] Chin I., Qin S.J., Lee K.S., Cho M.: "A two-stage iterative learning control technique combined with real-time feedback for independent disturbance rejection", Automatica, Vol. 40, No. 11, pp. 1913-1922, 2004

[52] Cho M., Lee Y., Joo S., Lee K.S.: "Semi-Empirical Model-Based Multivariable Iterative Learning Control of an RTP System", IEEE Transactions on Semiconductor Manufacturing, Vol. 18, No. 3, pp. 430-439, 2005

[53] Rice J.K., Verhaegen M.: "Lifted repetitive learning control for stochastic LTV systems: A structured matrix approach", Submitted to: Automatica, March, 2007

[54] de Roover D., Bosgra O.H.: "Synthesis of robust multivariable iterative learning controllers with application to a wafer stage motion system", Int. J. Contr., vol. 73, no. 10, pp. 968-979, 2000

[55] Mezghani M., Roux G., Cabassud M., Le Lann M.V., Dahhou B., Casamatta G.: "Application of iterative learning control to an exothermic semibatch chemical reactor", IEEE Trans. Contr. Syst. Technol., vol. 10, no. 6, pp. 822-834, 2002

[56] Norrlof M.: "An adaptive iterative learning control algorithm with experiments on an industrial robot", IEEE Trans. Robot. Automat., vol. 18, no. 2, pp. 245-251, 2002

[57] Mahony R., Hamel T.: "Adaptive compensation of aerodynamic effects during takeoff and landing manoeuvres for a scale model autonomous helicopter", European Journal of Control (EJC), Vol. 7, No. 1, pp. 43-58, 2001

[58] Ghosh J., Paden B.: "Pseudo-inverse based iterative learning control for nonlinear plants with disturbances", Proceedings of the 38th IEEE Conference on Decision and Control, Vol. 5, pp. 5206-5212, 1999, DOI 10.1109/CDC.1999.833379

[59] Moore A.: "Acquisition of Dynamic Control Knowledge for a Robotic Manipulator", Proceedings of the 7th International Conference on Machine Learning, Austin, Texas, United States, 1990, pp. 244-252

[60] Nice E.B.: "Design of Four Rotor Hovering Vehicle", Thesis presented for the degree of Master of Science, Cornell University, May 2004

[61] Schwartz A., Polak E., Chen Y.: "RIOTS 95", Optimal Control Toolbox for Matlab V6.5

[62] http://www.inertialscience.com/isis_imu.htm

[63] http://www.vicon.com/

[64] http://control.mae.cornell.edu/Purwin/PhDWork.htm